MARS: Enabling Verifiable Range-Aggregate Queries in Multi-Source Environments

Qin Liu, *Member, IEEE,* Yu Peng, *Graduate Student Member, IEEE,* Qian Xu, *Graduate Student Member, IEEE,* Hongbo Jiang, *Senior Member, IEEE,* Jie Wu, *Fellow, IEEE,* Tian Wang, *Member, IEEE,* Tao Peng, *Member, IEEE,* Guojun Wang, *Member, IEEE,* and Shaobo Zhang, *Member, IEEE*

Abstract—The huge values created by big data and the recent advances in cloud computing have been driving data from different sources into cloud repositories for comprehensive query services. However, cloud-based data fusion makes it challenging to verify if an untrusted server faithfully integrates data and executes queries or not. This is even harder for range-aggregate queries that apply aggregate operations on data within given ranges. In this paper, we propose a query authentication scheme, named MARS, enabling a user to efficiently authenticate range-aggregate queries on multi-source data. Specifically, MARS creates a VG-tree by subtly integrating Expressive Set Accumulator into a multi-dimensional G-tree while signing the root digest with a multi-source aggregate signature scheme. Compared with previous solutions, MARS has the following merits: (1) *Practicality.* Instead of treating range and aggregate queries separately, the user can directly verify the statistical result of selected data. (2) *Scalability.* Instead of authenticating the individual result from each source, the user can perform an aggregative validation on the integrated result from multiple sources. The experimental results demonstrate the effectiveness of MARS. For large-scale data fusion, the user-side verification time increases by only 103ms as the amount of data sources increases by five times.

Index Terms—Cloud computing, data fusion, authentication, multi-source data, range-aggregate queries

1 Introduction

In recent years, data fusion that integrates data from multiple sources and provides users with united analysis results has become a common trend in big data applications [1]. For example, sentiment analysis systems fuse multi-source corpora data to make a more accurate prediction [2]; intelligent transportation systems combine multi-source traffic data to preferably monitor road network [3]. As the increasing number of data sources producing enormous data flows, maintaining in-house data fusion infrastructure may incur expensive overheads. For cost-effectiveness, entrusting cloud service providers (CSPs) to merge data and provide query services has emerged as a more popular choice [4].

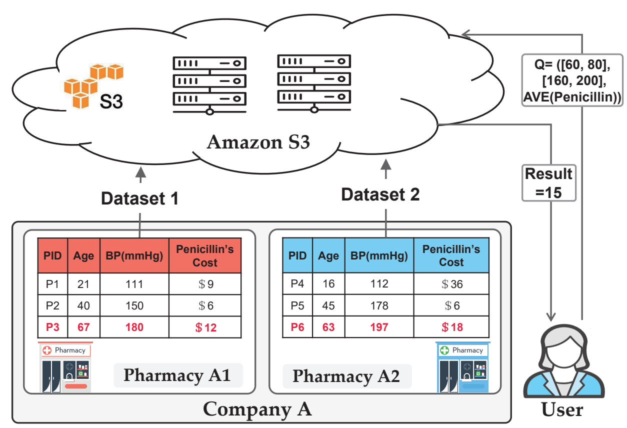


Fig. 1: Cloud-based medical data fusion.

Let us consider an application shown in Fig. 1. For better analysis of drug consumption trend, chain pharmacies of Company A cooperate to build a united medical data system relying on Amazon S3 [5]. Once authorized, a user is able to obtain the statistical results of selected data by querying the cloud-based data fusion system. For example, a staff of Company A can issue a query $\mathcal{Q} = (Age = [60, 80], BP = [160, 200], AVE(Penicillin))$ to get the average expense on

- *Qin Liu, Yu Peng, Qian Xu, and Hongbo Jiang are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan Province 410082, China. E-mail: {gracelq628, pengyu411, kilometerxu}@hnu.edu.cn; hongbojiang2004@gmail.com*
- *Jie Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA. E-mail: jiewu@temple.edu*
- *Tian Wang is with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University & UIC, Zhuhai, Guangdong Province 519000, China. E-mail: cs_tianwang@163.com*
- *Tao Peng and Guojun Wang are with the School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou, Guangdong Province 510006, China. E-mail: {pengtao, csgjwang}@gzhu.edu.cn*
- *Shaobo Zhang is with the School of Computer Science and Engineering of the Hunan University of Science and Technology, Xiangtan, Hunan Province 411201, China. E-mail: shaobozhang@hnust.edu.cn*

This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFE0201400, in part by the NSFC Grants 62272150, U20A20181, 62272162, 62172159, and 61872133, in part by the Humanities and Social Sciences Research Project of the Ministry of Education of China under Grant 22YJAZH155, in part by the Natural Science Foundation of Guangdong Province of China under Grant 2023A1515012358; and in part by the Hunan Provincial Natural Science Foundation of China under Grants 2021JJ30294, 2023JJ30267, 2020JJ3015. (Corresponding authors: Yu Peng and Tao Peng.)

Penicillin for the patients with age between 60 and 80 and blood pressure between 160 and 200.

Being affected by various factors inside and outside (e.g., external attacks, internal misconfiguration errors, software bugs, and insider threats), a CSP may return incorrect and incomplete results consciously or unconsciously. For example, BlueBleed leaked sensitive data of $65,000$+ entities in 111 countries because of a single misconfigured data bucket in Azure[1]; FlexBooker suffered a data breach exposing information of 3.7 million users due to a compromised account within Amazon web service infrastructure[2]. Returning to our application scenario, Amazon may return 12 instead of 15 as the result owing to the accidental loss of record P6.

To ensure result authenticity, existing studies [6]–[15] suggest the data owner to sign an authenticated data structure (ADS) so that the CSP can construct a verifiable object (VO) for the user to authenticate query results. However, previous verifiable query solutions mainly focus on a single data source without considering combing range and aggregate queries for unified verification, and thus they are unable to accommodate the requirements in the above application: (1) *Verifiable range-aggregate queries.* The statistical result of selected data serves as the basis of data analysis. The user should be able to authenticate aggregate operations (e.g., *AVE*, *SUM*, and *COUNT*) on the data within the given query ranges. (2) *Data integration*. The incompleteness of data sources makes data analysis meaningless. The user should be able to verify if the CSP faithfully fuses data or not. (3) *Efficiency.* The verification process should be lightweight enough that the user can verify the result anytime and anywhere even if using resource-constrained devices.

To realize the above requirements, this paper proposes a multi-source authenticated range-aggregate scheme, MARS, based on a VG-tree-based ADS and a well-designed multi-source aggregate signature (MSAS) scheme. Specifically, the VG-tree is constructed by subtly integrating Expressive Set Accumulator (ESA) [18] into a multi-dimensional G-tree [8], [9]; while the MSAS scheme is constructed by incorporating secret sharing scheme [20] into an improved identity-based aggregate signature (IBAS) scheme [21]. The main trick is that both the VG-tree and signature have the property of *combinableness*, enabling the CSP to construct an integrated VO and an aggregate signature based on multi-source fused data, so that the user can perform the *aggregative validation* in a lightweight way. To better understand the benefits of aggregative validation, we provide two constructions, MARS^0 and MARS^+. As the basic construction, MARS^0 let the CSP construct an individual VO based on each ADS for the user to authenticate range-aggregate queries on each source separately, while the advanced construction MARS^+ allows the CSP to return an integrated VO based on a merged ADS for the user to execute one-time verification. Detailed discussions are also provided to support dynamic rich queries and improve system practicality. The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work to devise an aggregative validation solution for range-aggregate queries in a multi-source environment.
- Based on the VG-tree and the MSAS scheme, MARS allows a user to authenticate range-aggregate queries on multi-source data in an efficient way. Compared with previous solutions, MARS has the following advantages: (1) *Practicality*. The user can authenticate the statistical results for the data selected as needed. (2) *Scalability*. The user can perform an aggregative validation on multi-source fused data at once.
- We conduct formal security analyses and an empirical study to validate the effectiveness of MARS. At worst, it needs only around 160ms for the user to verify range-aggregate queries on a united dataset consisting of millions of records.

1. https://socradar.io/details-on-the-largest-b2b-leak-bluebleed/
2. https://securityaffairs.com/126409/data-breach/flexbooker-data-breach.html

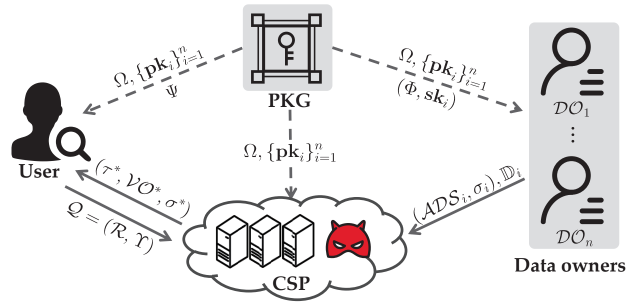


Fig. 2: System model. The communication channels between all entities are assumed to be secured under SSL/TLS.

Paper Organization. Section 2 formulates the problem, followed by the description of the basic construction in Section 3. After describing the advanced construction in Section 4, we provide analyses and discussions in Sections 5 and 6, respectively. We evaluate the constructions in Section 7 before introducing the related work in Section 8 Finally, we conclude this paper in Section 9.

2 Problem Formulation

2.1 The System and Threat Models

As shown in Fig. 2, our system model consists of four types of entities: n data owners/sources, denoted by $\mathcal{DO}_1, \ldots, \mathcal{DO}_n$, an authorized user, a CSP, and a private key generator (PKG). A data owner $\mathcal{DO}_i$ possesses a dataset $\mathbb{D}_i$, where the data is modeled as a set of n_i objects $\{o_1, \ldots, o_{n_i}\}$. To produce comprehensive analysis results, the n data owners cooperate to build a united dataset $\mathbb{D}^* = \bigcup_{i=1}^n \mathbb{D}_i$, and prefer to upload their own datasets to the CSP for ease of data fusion. Each object $o_k \in \mathbb{D}^*$ is represented as $(\mathbf{A}_k, v_k)$, where $\mathbf{A}_k$ is a vector of d comparative attribute values, and v_k is a functional attribute value[3]. To enable verifiable queries, $\mathcal{DO}_i$ also outsources an authenticated structure signed by its signature $(\mathcal{ADS}_i, \sigma_i)$.

The user issues range-aggregate queries to obtain the statistical results of selected data. A query is in the form of $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, where $\mathcal{R}$ is a d-dimensional range query on comparative attributes, and Υ is an aggregate operator on the functional attribute. This work mainly considers four kinds of aggregate operators: *SUM*, *COUNT*,

3. The case of multiple functional attribute values can be handled similarly and will not be described here for simplicity.

MIN, and *MAX*. For example, for $\mathcal{Q}_1 = (Age = [10, 30], BP = [100, 120]), SUM(Penicillin))$ and $\mathcal{Q}_2 = (Age = [60, 80], BP = [160, 200]), MIN(Penicillin))$, the results from the example dataset shown in Fig. 1 are 45 and 12, respectively.

The CSP pools massive resources to provide verifiable range-aggregate query services over the united dataset $\mathbb{D}^*$. Once receiving a query request $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, the CSP first executes the range query $\mathcal{R}$ on dataset $\mathbb{D}^*$ to screen out candidate objects, and then performs the aggregate operation Υ on these objects to obtain the statistical result τ^*. To enable the user to validate the final result τ^*, the CSP also returns an integrated VO and an aggregated signature $(\mathcal{VO}^*, \sigma^*)$.

Corresponding to the scenario of Fig. 1, chain pharmacies of Company A are data owners, the staff of Company A is the user, and Amazon is the CSP. Besides, our system also includes a PKG responsible for broadcasting system parameters and public keys $(\Omega, \{\mathbf{pk}_i\}_{i=1}^n)$, distributing secret keys $(\Phi, \mathbf{sk}_i)$ to $\mathcal{DO}_i$, and sending auxiliary message Ψ to the user. Unlike the CSP executing resource-intensive tasks, the PKG performs only lightweight operations during system initialization and can be maintained in-house. For instance, an edge server deployed in Company A may act as the PKG.

Following previous work [29], [32], [34], our threat model assumes that the data owners, the user, and the PKG are trustworthy, and assumes the CSP as a potential adversary may return incorrect statistical results by design or accident. For a query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, the user verifies if the CSP faithfully executes the query in the following aspects:

- **Integrity of Range Query**. No object in the query range $\mathcal{R}$ is overlooked or tampered with.
- **Correctness of Aggregate Query**. The statistical result of the objects in the query range $\mathcal{R}$ is authentic.
- **Completeness of Data Sources**. No data source is skipped in the query process.

2.2 Notations

Let $\lambda \in \mathbb{N}$ be a security parameter throughout this paper. Notation $[n]$ represents the set of integers $\{1, \ldots, n\}$. For a finite set $X = \{x_1, \ldots, x_n\}$, notation $|X|$ denotes its cardinality. The set of all binary strings of length x is denoted by $\{0, 1\}^x$ and the set of finite binary strings is denoted by $\{0, 1\}^*$. Notation $\|$ denotes string concatenation.

The outsourced dataset $\mathbb{D}^* = \bigcup_{i=1}^n \mathbb{D}_i$ comes from n data owners. $\mathcal{DO}_i$ with public/private key pair $(\mathbf{pk}_i, \mathbf{sk}_i)$ creates a VG-tree $\mathcal{VGT}_i$ as the ADS and produces a signature σ_i. In VG-tree $\mathcal{VGT}_i$, the node $\mathcal{VN}_{i,x}$ with label x is composed of a G-tree node $\mathcal{N}_{i,x}$ and a digest $\delta_{i,x}$. After merging n ADSs and n signatures, the CSP produces an integrated VG-tree $\mathcal{VGT}^*$ and an aggregate signature σ^*. For quick reference, the most relevant notations are shown in Table 1.

2.3 Cryptographic Preliminaries

Bilinear Pairing [17]. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups of prime order p with g being a generator of group $\mathbb{G}$. A bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ has the following properties: (1)*Bilinearity*: $\hat{e}(h^x, u^y) = \hat{e}(h, u)^{x \cdot y}$ for all $h, u \in \mathbb{G}$ and $x, y \in \mathbb{Z}_p$. (2)*Non-degeneracy*: $\hat{e}(g, g) \neq 1$. A bilinear-pairing parameter generator is denoted by $(p, \mathbb{G}, \mathbb{G}_T, \hat{e}, g) \leftarrow \mathcal{BG}(\lambda)$.

TABLE 1: Summary of notations

Notations	Descriptions
$(\mathbf{pk}_i, \mathbf{sk}_i)$	The public/private key pair of $\mathcal{DO}_i$
$(\mathcal{VGT}_i, \sigma_i)$	The VG-tree and signature created by $\mathcal{DO}_i$
$(\mathcal{VGT}^*, \sigma^*)$	The integrated VG-tree and aggregated signature
$\mathcal{N}_{i,x}, \delta_{i,x}$	The node with label x and its digest in tree $\mathcal{VGT}_i$
$\mathcal{N}_x^*, \delta_x^*$	The node with label x and its digest in tree $\mathcal{VGT}^*$
$\mathsf{OBJ}_{i,x}$	The set of objects included in node $\mathcal{N}_{i,x}$
$V_{i,x}$	The functional attribute values of objects in $\mathsf{OBJ}_{i,x}$
$\Upsilon(X)$	Execute aggregate operation Υ on set X
$\mathbf{acc}_X$	The accumulative value of set X

Expressive Set Accumulator [18]. ESA provides a succinct digest for a large set and a constant-size witness for various set operations. Due to its expressiveness, ESA is employed to authenticate aggregate operations. Let $[q]$ denote the universal functional attribute values, where $q = poly(\lambda)$, and let $\mathcal{A}_V(x) = \sum_{v \in V} x^v$ be a polynomial on set V. The tailored ESA scheme consists of the following algorithms:

• $(\Omega_E, \alpha) \leftarrow \mathsf{Setup}(\lambda)$: The PKG picks a random secret key $\alpha \in \mathbb{Z}_p^*$, gets $\mathbf{pub} = (p, \mathbb{G}, \mathbb{G}_T, \hat{e}, g)$ by running $\mathcal{BG}(\lambda)$, and sets the public parameters as $\Omega_E = \{\mathbf{pub}, \{g^{\alpha^v}\}_{v \in [q]}\}$.

• $\mathbf{acc}_V \leftarrow \mathsf{GenAcc}(V, \Omega_E)$: The data owner defines a polynomial $\mathcal{A}_V(x)$ and calculates the accumulative value for set V as $\mathbf{acc}_V = g^{\mathcal{A}_V(\alpha)} = g^{\sum_{v \in V} \alpha^v}$.

• $(\pi, \tau) \leftarrow \mathsf{GenProof}(V, \Upsilon, \Omega_E)$: The CSP defines a polynomial $\mathcal{A}_V(x)$ and performs in terms of operator Υ[4]:

(1) *COUNT* : It calculates $\mathcal{B}_V(x) = \frac{\mathcal{A}_V(x) - \mathcal{A}_V(1)}{x-1}$, and sets $\tau = \mathcal{A}_V(1)$ and $\pi = g^{\mathcal{B}_V(\alpha)}$. (2) *SUM* : It calculates $\mathcal{B}_V(x) = \frac{\mathcal{A}_V(x) - \mathcal{A}_V(1) - \mathcal{A}'_V(1)(x-1)}{(x-1)^2}$, and sets $\tau = \mathcal{A}'_V(1)$ and $\pi = (g^{\mathcal{B}_V(\alpha)}, \mathcal{A}_V(1))$, where $\mathcal{A}'(x)$ is the derivative of $\mathcal{A}(x)$. (3) *MIN*: It sets $\tau = \min(V)$ and $\pi = g^{(\mathcal{A}_V(\alpha) - \alpha^\tau)/\alpha^{\tau+1}}$.

• $\{0, 1\} \leftarrow \mathsf{VerProof}(\mathbf{acc}_V, \Upsilon, \pi, \tau, \Omega_E)$: To verify the result τ, the user performs in terms of operator Υ:

(1) *COUNT* : It outputs 1 iff $\hat{e}(\mathbf{acc}_V / g^\tau, g) = \hat{e}(g^{\alpha - 1}, \pi)$. (2) *SUM* : It parses π as $(\pi_1, \pi_2) = (g^{\mathcal{B}_V(\alpha)}, \mathcal{A}_V(1))$ and outputs 1 iff $\hat{e}(\mathbf{acc}_V, g) = \hat{e}(g^{(\alpha-1)^2}, \pi_1) \cdot \hat{e}(g^{\tau \cdot (\alpha - 1) + \pi_2}, g)$. (3) *MIN* : It outputs 1 iff $\hat{e}(\mathbf{acc}_V, g) = \hat{e}(g^{\alpha^\tau}, g) \cdot \hat{e}(\pi, g^{\alpha^{\tau+1}})$.

The security of ESA is based on q-SBDH assumption [19]. That is, given the parameters Ω_E, the aggregate operator Υ, and the accumulative value $\mathbf{acc}_V$, it is difficult to find $\tau' \neq \Upsilon(V)$ and π' such that $1 \leftarrow \mathsf{VerProof}(\mathbf{acc}_V, \Upsilon, \pi', \tau', \Omega_E)$.

***(n, n)*-Secret-Sharing Scheme [20].** It divides a secret s into n shares that are securely shared among n parties, such that only all n parties together can reconstruct the secret s. To implement, the first $n-1$ parties create random values $ss_1, \ldots, ss_{n-1}$ and the last party sets ss_n to $s - \sum_{i=1}^{n-1} ss_i$. Therefore, the secret can be obtained by $s = \sum_{i=1}^n ss_i$.

Identity-based Aggregate Signature [21]. IBAS enables to aggregate n parties' signatures on n distinct messages into a compact signature for one-time validation. Its security is based on CDH assumption [22] and one-time-use disturbances. The tailored IBAS scheme works as follows.

• $(\Omega_I, \beta) \leftarrow \mathsf{Setup}(\lambda)$: The PKG randomly chooses a master key $\beta \in \mathbb{Z}_p$, and sets the public parameters as $\Omega_I =$

4. To reduce the parameter size from $O(q^2)$ to $O(q)$, operator *MAX* is realized by operator *MIN* after replacing each element $v \in V$ with $q - v$. Besides, operator *AVE* can be realized by *SUM* and *COUNT* operators.

$(\mathbf{pub}, h, H_1)$, where $\mathbf{pub} = (p, \mathbb{G}, \mathbb{G}_T, \hat{e}, g) \leftarrow \mathcal{BG}(\lambda)$, $h = g^{\beta}$, and $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$ is a cryptographic hash function.

- $(\mathbf{pk}_i, \mathbf{sk}_i) \leftarrow \mathsf{GenKey}(\mathcal{ID}_i, \beta, \Omega_I)$: The PKG calculates the public/private key pair for $\mathcal{DO}_i$ with identity $\mathcal{ID}_i$ by setting $\mathbf{pk}_i = (h_{i,0}, h_{i,1})$ and $\mathbf{sk}_i = (h_{i,0}^{\beta}, h_{i,1}^{\beta})$, where $h_{i,0} \leftarrow H_1(\mathcal{ID}_i \| 0)$ and $h_{i,1} \leftarrow H_1(\mathcal{ID}_i \| 1)$.
- $\sigma_i \leftarrow \mathsf{GenSig}(m_i, \mathbf{sk}_i, \Omega_I)$: To sign a message $m_i \in \mathbb{Z}_p$, $\mathcal{DO}_i$ with $\mathbf{sk}_i = (h_{i,0}^{\beta}, h_{i,1}^{\beta})$ chooses a disturbance $\mathcal{W} \in \{0,1\}^*$ hasn't been used before and a random element $r_i \in \mathbb{Z}_p$. It generates a signature $\sigma_i = (\mathcal{W}, \omega_i, \varphi_i)$, where $\omega_i = H_1(\mathcal{W})^{r_i} \cdot h_{i,0}^{\beta} \cdot h_{i,1}^{\beta \cdot m_i}$ and $\varphi_i = g^{r_i}$.
- $\sigma^* \leftarrow \mathsf{AggSig}(\{\sigma_i\}_{i=1}^n, \Omega_I)$: The CSP aggregates n signatures $\{\sigma_i\}_{i=1}^n$ with the same disturbance $\mathcal{W}$ by setting $\sigma^* = (\mathcal{W}, \omega^*, \varphi^*)$, where $\omega^* = \prod_{i=1}^n \omega_i$ and $\varphi^* = \prod_{i=1}^n \varphi_i$.
- $\{0,1\} \leftarrow \mathsf{VerSig}(\sigma^*, \{m_i\}_{i=1}^n, \{\mathbf{pk}_i\}_{i=1}^n, \Omega_I)$: The user verifies the signature σ^* and outputs 1 iff Eq. 1 is satisfied.

$$\hat{e}(\omega^*, g) = \hat{e}(\varphi^*, H_1(\mathcal{W})) \cdot \hat{e}(h, \prod_{i=1}^{n} h_{i,0} \cdot \prod_{i=1}^{n} h_{i,1}^{m_i}). \quad (1)$$

The IBAS scheme is reasonably efficient, requiring only a constant number of bilinear pairings for verification. Furthermore, an individual signature σ_i can be authenticated by running $\mathsf{VerSig}(\sigma_i, m_i, \mathbf{pk}_i, \Omega_I)$ without aggregation.

3 The Basic Construction

In the basic construction MARS^0, each data owner independently constructs a VG-tree as the ADS based on which the CSP generates an individual VO without merging data. Given n VO/signature pairs, the user authenticates the range-aggregate query over each dataset in turn.

Step 1: The PKG runs algorithm Initialization to generate the secret keys $\{\alpha, \beta\}$, the system parameters Ω, and the key pairs $(\mathbf{pk}_i, \mathbf{sk}_i)_{i=1}^n$. Here, $\{\alpha, \beta\}$ will be kept secret, $(\{\mathbf{pk}_i\}_{i=1}^n, \Omega)$ will be broadcast, and $\mathbf{sk}_i$ will be sent to $\mathcal{DO}_i$.

Step 2: $\mathcal{DO}_i$ runs algorithm ADS Generation to generate the VG-tree and signature $(\mathcal{VGT}_i, \sigma_i)$ that will be outsourced together with the dataset $\mathbb{D}_i$. After getting $(\mathbb{D}_i, \mathcal{VGT}_i, \sigma_i)_{i=1}^n$ from n data owners, the CSP keeps them separately without performing merging operation.

Step 3: On receiving a query $\mathcal{Q}$ from the user, the CSP runs algorithm VO Construction to get intermediate results and VOs $(\tau_i, \mathcal{VO}_i)_{i=1}^n$ from n datasets $\{\mathbb{D}_i\}_{i=1}^n$ and returns them along with data owners' signatures $\{\sigma_i\}_{i=1}^n$.

Step 4: The user runs the Verification algorithm to verify intermediate values $\tau_1, \ldots, \tau_n$ in sequence. If all validations pass, the final result τ^* is calculated as $\Upsilon(\{\tau_1, \ldots, \tau_n\})$.

3.1 VG-Tree

To construct an ADS for verifiable range-aggregate queries, our main idea is first to construct a d-dimensional G-tree from comparative attribute values, and then extending it to a VG-tree by incorporating a digest into each G-tree node.

G-tree Construction. Let $[L_k, U_k]$ denote the value range of the k-th comparative attribute, for $k \in [d]$. A G-tree $\mathcal{GT}$ is a 2^d-ary tree, where each node $\mathcal{N}_x$ is defined as follows:

Definition 1 (G-tree node)**.** *If $\mathcal{N}_x$ is a non-leaf node, $\mathcal{N}_x = \langle x, gb_x, x_1, \ldots, x_{2^d} \rangle$; otherwise, $\mathcal{N}_x = \langle x, gb_x, \mathsf{OBJ}_x \rangle$, where x is the node label, gb_x is the d-dimensional grid associated with $\mathcal{N}_x$, x_i is the label of i-th children of node $\mathcal{N}_x$, and OBJ_x is the identifiers of objects included in this node.*

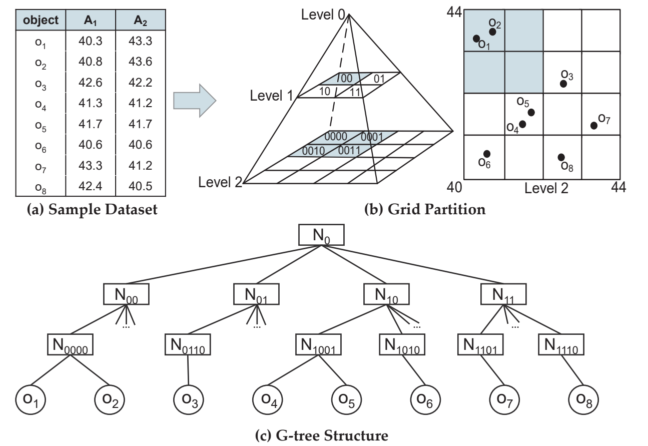


Fig. 3: An example of 2-dimensional G-tree. The leaf nodes with no object included are omitted. The value range of each dimension is set as: $[L_1, U_1] = [L_2, U_2] = [40.0, 44.0]$.

Each tree node is labeled with prefix code [23]. Taking $[L_k, U_k]_{k=1}^d$ as the grid associated with the root node, the grids at next levels are formed as follows: For each grid at the i-th level, it is divided into 2^d equal-size sub-grids to form the grids at the $(i+1)$-th level. The partition is performed in a recursive manner until the number of grids at the current level reaches a predefined threshold. Fig. 3 shows an example 2-dimensional G-tree. Initially, the root grid is $gb_0 = \{[40.0, 44.0], [40.0, 44.0]\}$, which is partitioned into 4 sub-grids of the same size, $\{gb_{00}, gb_{01}, gb_{10}, gb_{11}\}$. If the threshold is set to 16, the tree height is 3.

VG-tree Construction. Let $H : \{0,1\}^* \rightarrow \{0,1\}^{\lambda}$ be a cryptographic hash function. A d-dimensional G-tree $\mathcal{GT}$ can be extended to a VG-tree $\mathcal{VGT}$ by incorporating a digest into each tree node from the bottom up as follows:

Definition 2 (Leaf node digest)**.** *For a leaf node $\mathcal{VN}_x = \langle \mathcal{N}_x, \delta_x \rangle$, where $\mathcal{N}_x = \langle x, gb_x, \mathsf{OBJ}_x \rangle$ is a G-tree leaf node, the digest δ_x is calculated by Eq. 2:*

$$\delta_x = H(gb_x || H(\mathbf{acc}_{V_x})), \quad (2)$$

where V_x is the functional values of objects in OBJ_x, and $\mathbf{acc}_{V_x}$ is the accumulative value of set V_x. In a special case, if $\mathsf{OBJ}_x = \emptyset$, $\mathbf{acc}_{V_x}$ is set to $\perp$ and $H(\mathbf{acc}_{V_x})$ is set to a dummy string $\mathcal{S}$.

Definition 3 (Non-leaf node digest)**.** *For a non-leaf node $\mathcal{VN}_x = \langle \mathcal{N}_x, \delta_x \rangle$, where $\mathcal{N}_x = \langle x, gb_x, x_1, \ldots, x_{2^d} \rangle$ is a G-tree non-leaf node, the digest δ_x is calculated by Eq. 3:*

$$\delta_x = H(gb_{x_1} || \delta_{x_1} || \ldots || gb_{x_{2^d}} || \delta_{x_{2^d}}). \quad (3)$$

Each data owner independently builds trees from its own dataset. In order to distinguish them, the G-tree and VG-tree created by $\mathcal{DO}_i$ are denoted by $\mathcal{GT}_i$ and $\mathcal{VGT}_i$, where the node with label x is denoted by $\mathcal{N}_{i,x}$ and $\mathcal{VN}_{i,x} = (\mathcal{N}_{i,x}, \delta_{i,x})$, respectively. In particular, for a non-leaf node, $\mathcal{N}_{i,x}$ is in the form of $(x, gb_{i,x}, (i, x_1), \ldots, (i, x_{2^d}))$, while for a leaf node $\mathcal{N}_{i,x}$ is in the form of $(x, gb_{i,x}, \mathsf{OBJ}_{i,x})$.

3.2 Details of MARS^0

Let ESA and IBAS be an ESA scheme and an IBAS scheme described in Section 2.3, respectively. The details of MARS^0

Algorithm 1 Basic Construction MARS^0

Initialization (by the PKG)

Input: Security parameter λ, data owners' identities $\{\mathcal{ID}_i\}_{i=1}^n$
Output: Parameter Ω, keys $\{\alpha, \beta\}$, key pairs $(\mathbf{pk}_i, \mathbf{sk}_i)_{i=1}^n$
$(\Omega_E, \alpha) \leftarrow$ ESA.Setup(λ); $(\Omega_I, \beta) \leftarrow$ IBAS.Setup(λ)
for $i = 1$ **to** n **do**
  $(\mathbf{pk}_i, \mathbf{sk}_i) \leftarrow$ IBAS.GenKey($\mathcal{ID}_i, \beta, \Omega_I$)
$\Omega \leftarrow (\Omega_E, \Omega_I)$

ADS Generation (by $\mathcal{DO}_i$)

Input: Dataset $\mathbb{D}_i$, public/private key pair $(\mathbf{pk}_i, \mathbf{sk}_i)$
Output: The ADS $\mathcal{VGT}_i$ and signature σ_i
Construct a d-dimensional G-tree $\mathcal{GT}_i$ according to Def. 1
for each leaf node of $\mathcal{N}_{i,x} = (x, gb_{i,x}, \mathsf{OBJ}_{i,x})$ **do**
  $V_{i,x} \leftarrow \{v_k\}_{o_k \in \mathsf{OBJ}_{i,x}}$; $\mathbf{acc}_{V_{i,x}} \leftarrow$ ESA.GenAcc($V_{i,x}, \Omega_E$)
Extend $\mathcal{GT}_i$ to a VG-tree $\mathcal{VGT}_i$ based on Def. 2 and Def. 3
$\sigma_i \leftarrow$ IBAS.GenSig($\delta_{i,0}, \mathbf{sk}_i, \Omega_I$)

VO Construction (by the CSP)

Input: Query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, datasets $\{\mathbb{D}_i\}_{i=1}^n$, ADSs $\{\mathcal{VGT}_i\}_{i=1}^n$
Output: Intermediate results $\{\tau_i\}_{i=1}^n$, VOs $\{\mathcal{VO}_i\}_{i=1}^n$
for $i = 1$ **to** n **do**
  Run Query($\mathcal{VGT}_i.root, \mathbb{D}_i, \mathcal{R}$) to output $(V_i, \mathrm{MN}_i, \mathrm{UN}_i)$
  $(\pi_i, \tau_i) \leftarrow$ ESA.GenProof(V_i, Υ, Ω_E)
  $\mathcal{VO}_i \leftarrow (\mathrm{MN}_i, \mathrm{UN}_i, \pi_i)$

Verification (by the user)

Input: The query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, VOs/signatures $\{\mathcal{VO}_i, \sigma_i\}_{i=1}^n$, intermediate results $\{\tau_i\}_{i=1}^n$, public keys $\{\mathbf{pk}_i\}_{i=1}^n$
Output: Verification report $\mathcal{VR}$, final result τ^*
$c \leftarrow 0$ ▷ *c denotes the number of verified intermediate results*
for $i = 1$ **to** n **do**
  Pares $\mathcal{VO}_i$ as $\{\mathrm{MN}_i, \mathrm{UN}_i, \pi_i\}$
  if MN_i and UN_i pass validation **then**
    Reconstruct $\mathcal{VGT}_i$ and $\delta_{i,0}$ with Def. 2 and Def. 3
    if IBAS.VerSig($\sigma_i, \delta_{i,0}, \mathbf{pk}_i, \Omega_I$) **then**
      $\mathbf{acc}_{V_i} \leftarrow \prod_{\mathcal{VN}_{i,x} \in \mathrm{MN}_i} \mathbf{acc}_{V_{i,x}}$
      if ESA.VerProof($\mathbf{acc}_{V_i}, \Upsilon, \pi_i, \tau_i, \Omega_E$) **then**
        $c \leftarrow c + 1$
if $c \neq n$ **then**
  $\mathcal{VR} \leftarrow 0$; $\tau^* \leftarrow \perp$ ▷ *0 indicates verification fails*
else
  $\mathcal{VR} \leftarrow 1$; $\tau^* \leftarrow \Upsilon(\{\tau_1, \ldots, \tau_n\})$

Algorithm 2 Query (the boxed codes are for MARS$^+$)

Input: VG-tree $\mathcal{VGT}.root$, dataset $\mathbb{D}$, range query $\mathcal{R}$
Output: Functional values V, set MN, set UN
$\mathbf{Q} \leftarrow$ empty queue; $(\mathrm{V}, \mathrm{MN}, \mathrm{UN}) \leftarrow$ empty set
Push $\mathcal{VRT}.root$ into queue $\mathbf{Q}$
while $\mathbf{Q}$ is non-empty **do**
  $\mathcal{VN}_x \leftarrow$ the head of queue $\mathbf{Q}$
  if $\mathcal{VN}_x$ is a non-leaf node **then**
    if gb_x disjoints from $\mathcal{R}$ **then**
      Put (x, gb_x, δ_x) into UN
    else
      Push all the children nodes of $\mathcal{VN}_x$ into queue $\mathbf{Q}$
  if $\mathcal{VN}_x$ is a leaf node **then**
    if gb_x is not included in $\mathcal{R}$ **then**
      Put $\{x, gb_x, \mathbf{acc}_{V_x}\}$ into UN
      $\boxed{\text{Put } (x, gb_x, \{\mathbf{acc}_{V_{i,x}}\}_{i=1}^n) \text{ into UN}}$
    else
      Put $(x, gb_x, \mathbf{acc}_{V_x})$ into MN
      ~~$V \leftarrow V \cup V_x$~~
      $\boxed{\begin{array}{l}\text{Put } (x, gb_x, \{\mathbf{acc}_{V_{i,x}}\}_{i=1}^n) \text{ into MN} \\ V \leftarrow V \cup \bigcup_{i=1}^n V_{i,x}\end{array}}$

are shown in Alg. 1, where every algorithm but Initialization takes the system parameters Ω as the implicit input.

Initialization. The PKG first runs the ESA.Setup and IBAS.Setup algorithms to create system parameters Ω and secret keys $\{\alpha, \beta\}$, and then runs algorithm IBAS.GenKey to generate the key pair $(\mathbf{pk}_i, \mathbf{sk}_i)$ for $\mathcal{DO}_i$ ($i \in [n]$).

ADS Generation. $\mathcal{DO}_i$ first creates a G-Tree $\mathcal{GT}_i$ from dataset $\mathbb{D}_i$ and extends it to a VG-tree $\mathcal{VGT}_i$ by incorporating a digest into each node, where the accumulative value is calculated by algorithm ESA.GenAcc. Then, $\mathcal{DO}_i$ runs algorithm IBAS.GenSig to get a signature σ_i for the root digest $\delta_{i,0}$ (which is mapped to a value in $\mathbb{Z}_p$ before signing).

VO Construction. On receiving a query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, the CSP first runs algorithm Query on each VG-Tree $\mathcal{VGT}_i$ and dataset $\mathbb{D}_i$ to obtain the functional values V_i of candidate objects in query $\mathcal{R}$, a set of matched nodes MN_i, and a set of unmatched nodes UN_i. Then, the CSP runs algorithm ESA.GenProof to output the intermediate result $\tau_i = \Upsilon(V_i)$ and relevant proof π_i, and sets $\mathcal{VO}_i = (\mathrm{MN}_i, \mathrm{UN}_i, \pi_i)$.

The details of the query process are described in Alg. 2 (excluding the boxed codes). For each VG-tree $\mathcal{VGT}_i$, the CSP performs a detection starting from the root node $\mathcal{VRT}_i.root$ as follows: If a non-leaf node is disjoint with the range query $\mathcal{R}$, it stops traversing the subtree rooted at this node, and puts the label, the grid, and the digest into UN_i; Otherwise, it further checks all its children nodes. When this traversal reaches a leaf node, it puts the functional values of candidate objects into V_i, while putting the label, the grid, and relevant accumulative value into UN_i or MN_i depending on if the leaf is disjoint with query $\mathcal{R}$ or not. As a trade-off for time efficiency, the CSP may locally keep these accumulative values to avoid repeated calculations.

Verification. On receiving intermediate results $\{\tau_i\}_{i=1}^n$ and VO/signature pairs $(\mathcal{VO}_i, \sigma_i)_{i=1}^n$ from the CSP, the user performs the following three-stage verification to check if the CSP honestly executes the query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$ or not.

Stage 1. For each VO/signature pair $(\mathcal{VO}_i, \sigma_i)$, the user authenticates the integrity of range query $\mathcal{R}$ on dataset $\mathbb{D}_i$. It first examines if MN_i and UN_i in $\mathcal{VO}_i$ satisfy the following requirements: (1) For each $(x, gb_{i,x}, \mathbf{acc}_{V_{i,x}}) \in \mathrm{MN}_i$, $gb_{i,x}$ is included in query $\mathcal{R}$; (2) For each $(x, gb_{i,x}, *) \in \mathrm{UN}_i$, $gb_{i,x}$ disjoints with query $\mathcal{R}$. If so, it reconstructs the root digest $\delta_{i,0}$ of $\mathcal{VGT}_i$ with MN_i and UN_i and verifies the signature σ_i using algorithm IBAS.VerSig. Note that, the label and the grid of a parent node can be recovered from those of its children node according to the prefix encoding and the partitioning method of G-tree. Besides, it is hard for the CSP to falsify the labels and grids. This is because the G-tree structure is deterministic once the value ranges are predefined. If the validation passes, this means that all the components in MN_i and UN_i are indeed calculated from the original $\mathbb{D}_i$ and $\mathcal{VGT}_i$, and no candidate object is skipped, validating the integrity of the range query.

Stage 2. For each intermediate result τ_i, the user authenticates the correctness of the aggregate query Υ. It first calculates $\mathbf{acc}_{V_i} = \prod_{\mathcal{VN}_{i,x} \in \mathrm{MN}_i} \mathbf{acc}_{V_{i,x}}$ to obtain the accumulative value of candidate objects, and then runs algorithm ESA.VerProof to verify result τ_i. Due to the security of ESA, this algorithm outputs 1 only when $\tau_i = \Upsilon(V_i)$, validating the correctness of the aggregate query.

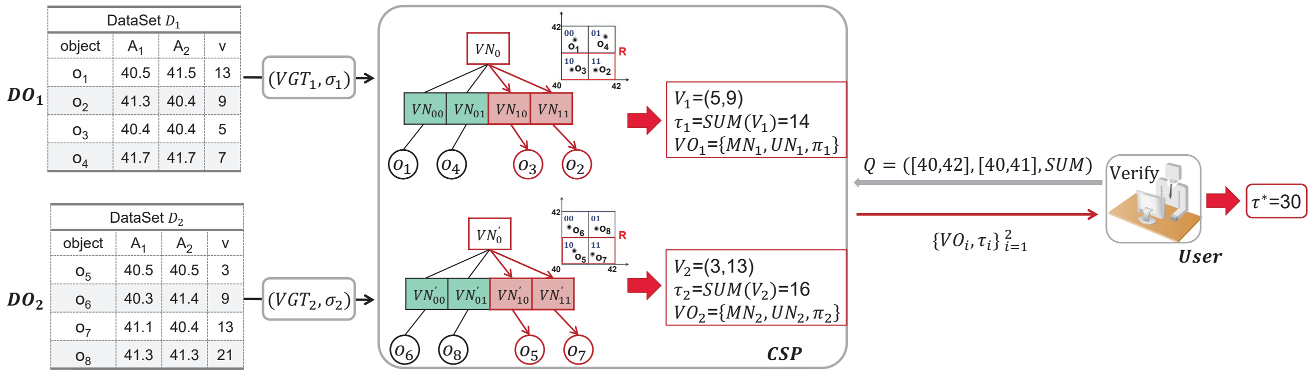


Fig. 4: Exemplary working process of MARS^0. As for a G-tree, the value range is $[40.0, 44.0]$ and the threshold is set to 4.

Stage 3. Once the correctness of n intermediate results is verified, the user calculates $\Upsilon(\{\tau_1, \ldots, \tau_n\})$ to obtain the final result τ^*, validating the completeness of data sources.

3.3 Illustrative Example

To illustrate the working process of MARS^0, let us consider the example shown in Fig. 4, where two data owners, $\mathcal{DO}_1$ and $\mathcal{DO}_2$, cooperate to build a unite dataset and a user issues a query $\mathcal{Q} = (\mathcal{R}, \mathit{SUM})$ to obtain the sum of functional values within range $\mathcal{R} = [40.0, 42.0], [40.0, 41.0]$.

ADS Generation. After system initialization, each $\mathcal{DO}_i$ independently creates a VG-tree, $\mathcal{VGT}_i$ of height 2, and then generates a signature σ_i for the root digest $\delta_{i,0}$.

VO Construction. For each dataset $\mathbb{D}_i$, the CSP runs algorithm Query on $\mathcal{VGT}_i$ to generate $(V_i, \mathrm{MN}_i, \mathrm{UN}_i)$. The search process upon the VG-trees is marked by red thick lines, while the matched and unmatched nodes are filled with red and green, respectively. For dataset $\mathbb{D}_1$, the candidate objects in $\mathcal{R}$ are $\{o_2, o_3\}$ with $V_1 = \{9, 5\}$. It runs algorithm ESA.GenProof to get π_1 and $\tau_1 = 14 \leftarrow \mathit{SUM}(V_1)$, and sets $\mathcal{VO}_1 = \{\mathrm{MN}_1, \mathrm{UN}_1, \pi_1\}$. For dataset $\mathbb{D}_2$, the candidate objects in $\mathcal{R}$ are $\{o_5, o_7\}$ with $V_2 = \{3, 13\}$. It runs algorithm ESA.GenProof to get π_2 and $\tau_2 = 16 \leftarrow \mathit{SUM}(V_2)$ and sets $\mathcal{VO}_2 = \{\mathrm{MN}_2, \mathrm{UN}_2, \pi_2\}$.

Verification. The user first reconstructs a VG-tree $\mathcal{VGT}_i$ based on each $\mathcal{VO}_i = \{\mathrm{UN}_i, \mathrm{MN}_i, \pi_i\}$, and verifies the root digest $\delta_{i,0}$ by algorithm IBAS.VerSig. For example, given $\mathrm{UN}_1 = \{(00, gb_{1,00}, \mathbf{acc}_{V_{1,00}}), (01, gb_{1,01}, \mathbf{acc}_{V_{1,01}})\}$ and $\mathrm{MN}_1 = \{(10, gb_{1,10}, \mathbf{acc}_{V_{1,10}}), (11, gb_{1,11}, \mathbf{acc}_{V_{1,11}})\}$, the user recover the structure of VG-tree $\mathcal{VGT}_1$ and calculates digests from the bottom up: After calculating the leaf digests $\delta_{1,00}$, $\delta_{1,01}$, $\delta_{1,10}$, and $\delta_{1,11}$ with Eq. 2, it calculates the root digest $\delta_{1,0}$ with Eq. 3. If the validation passes, it calculates the accumulative value $\mathbf{acc}_{V_i}$ for candidate objects in $\mathcal{VGT}_i$ and further verifies the intermediate result τ_i by algorithm ESA.VerProof. Once τ_1 and τ_2 are verified, it calculates $\mathit{SUM}(\{14, 16\})$ to obtain the final result $\tau^* = 30$.

4 The Advanced MARS Construction

MARS^0 supports multi-source range-aggregate queries in a verifiable way, but lacks scalability since both the VO construction and verification costs are linear with the number of data owners. To address this, MARS$^+$ designs a combinative digest signed by a MSAS scheme, so that the CSP can merge multi-source data to construct an integrated VO and an aggregate signature, enabling the user to perform aggregative verification in a lightweight way. The main differences from MARS^0 lie in the following aspects:

Step 1: The PKG runs algorithm Initialization to initialize the system and generate public/private keys to all data owners. Beyond that, it sends auxiliary messages Ψ to the user, and sends shared keys Φ to each data owner.

Step 2: $\mathcal{DO}_i$ runs algorithm ADS Generation to generate $(\mathcal{VGT}_i, \sigma_i)$ and outsources them together with dataset $\mathbb{D}_i$.

Step 3: Given $(\mathbb{D}_i, \mathcal{VGT}_i, \sigma_i)_{i=1}^n$, the CSP runs algorithm Merging to form an integrated VG-tree $\mathcal{VGT}^*$ for the united dataset $\mathbb{D}^*$ and produces an aggregated signature σ^*.

Step 4: Given a query $\mathcal{Q}$ from the user, the CSP runs algorithm VO Construction to produce the final result and integrated VO $(\tau^*, \mathcal{VO}^*)$ and returns them along with σ^*.

Step 5: With the integrated signature/VO pair $(\sigma^*, \mathcal{VO}^*)$ and the auxiliary messages Ψ, the user runs the Verification algorithm to verify the final result τ^* directly.

4.1 Main Idea

The G-tree is inherently mergeable when the value ranges of comparative attributes as well as the partitioning method are shared among data owners. Furthermore, the IBAS scheme as the basis of the MSAS scheme supports the aggregation of signatures. Therefore, our main idea is to make the digests *combinative*, while signing each root digest with the MSAS scheme, so that the CSP can produce an integrated VG-tree signed by an aggregated signature. However, this extension is non-trivial due to the following challenges:

Challenge 1: How to Design an Combinative Digest? Let Λ denote any merging operation. Given the digests $\delta_{1,x}, \ldots, \delta_{n,x}$ of n nodes with label x, the digest of the merged node is denoted by $\delta_x^* = \Lambda(\delta_{1,x}, \ldots, \delta_{n,x})$. To enable the user to reconstruct the integrated VG-tree, δ_x^* needs to be able to calculated from children nodes' digests $\delta_{x_1}^*, \ldots, \delta_{x_{2^d}}^*$. To fulfill this property, we set the digest $\delta_{i,x}$ to an element in $\mathbb{Z}_p$, s.t. $\delta_{i,x} = \sum_{k=1}^{2^d} \delta_{i,x_k} \mod p$, where $\{\delta_{i,x_k}\}_{k=1}^{2^d}$ are children nodes' digests. If Λ denotes the add operation in $\mathbb{Z}_p$, the combinative digest is obtained by: $\delta_x^* = \sum_{i=1}^n \delta_{i,x} = \sum_{i=1}^n \sum_{k=1}^{2^d} \delta_{i,x_k} = \sum_{k=1}^{2^d} \sum_{i=1}^n \delta_{i,x_k} = \sum_{k=1}^{2^d} \delta_{x_k}^*$.

Challenge 2: How to Combine with the MSAS Scheme? As described in Section 2.3, the IBAS.VerSig algorithm takes the root digests $\{\delta_{i,0}\}_{i=1}^n$ and the public keys $\{\mathbf{pk}_i\}_{i=1}^n$ as input to verify the aggregate signature σ^*. That is, the user needs to reconstruct n VG-trees $\{\mathcal{VGT}\}_{i=1}^n$ to obtain all the root digests $\{\delta_{i,0}\}_{i=1}^n$, before running this algorithm. To further reduce the user-side overhead, the MSAS scheme improves the IBAS scheme by subtly introducing a common share into data owners' public keys, so that the user can reconstruct an integrated VG-tree $\mathcal{VGT}^*$ and use its root digest δ_0^* straightway for efficient verification.

Challenge 3: How to Verify the Completeness of Data Sources? The verification algorithm examines digest authenticity, rather than the completeness of data sources. That is, algorithm IBAS.VerSig outputs 1 if δ_0^* is the integration of real root digests. To deceive the user, the CSP may merge partial VG-trees $\mathcal{VGT}_1, \ldots, \mathcal{VGT}_{n-1}$ to form $\widetilde{\mathcal{VGT}}^*$ while returning $\widetilde{\sigma}^* = \prod_{i=1}^{n-1} \sigma_i$ and $\widetilde{\mathcal{VO}}^*$ constructed from $\widetilde{\mathcal{VGT}}^*$. Since $\widetilde{\sigma}^*$ and $\widetilde{\mathcal{VO}}^*$ are indeed calculated from the original data, the CSP may successfully make the user accept $\widetilde{\delta}_0^*$ recalculated from $\widetilde{\mathcal{VO}}^*$. To solve this problem, we integrate the secret sharing scheme into the MSAS scheme so that algorithm MSAS.VerSig outputs 1 only when all the returned data is authentic and there is no skipped data owner.

4.2 The MSAS Scheme

The MSAS scheme built based on the IBAS scheme and secret sharing scheme consists of the following algorithms:

- $(\Omega_I, \beta) \leftarrow \mathsf{Setup}(\lambda)$: The PKG randomly chooses a master key $\beta \in \mathbb{Z}_p$ and sets the public parameters as $\Omega_I = (\mathbf{pub}, h, H_1, u)$, where u is a random element in $\mathbb{G}$ and the remaining components are defined in the same way as those in the IBAS.Setup algorithm.
- $(\mathbf{pk}_i, \mathbf{sk}_i) \leftarrow \mathsf{GenKey}(\mathcal{ID}_i, \beta, \Omega_I)$: The PKG chooses a random element $s_i \in \mathbb{Z}_p$ and calculates $h_{i,0} \leftarrow H_1(\mathcal{ID}_i \| 0)$ and $h_{i,1} = u^{s_i^{-1}}$, s.t. $h_{i,1}^{s_i} = u$. Then, it sets the public/private key pair as $\mathbf{pk}_i = (h_{i,0}, h_{i,1})$ and $\mathbf{sk}_i = (s_i, h_{i,0}^{\beta}, h_{i,1}^{\beta})$ for $\mathcal{DO}_i$ with identity $\mathcal{ID}_i$. Based on the CDH assumption, it is hard for an adversary to recover $\{s_i\}_{i=1}^n$ from $(u, \{h_{i,1}\}_{i=1}^n)$.
- $\sigma_i \leftarrow \mathsf{GenSig}(m_i, \mathbf{sk}_i, \Omega_I)$: Before signing, n data owners negotiate the disturbance $\mathcal{W} \in \{0,1\}^*$ and then collaboratively run SSS to get random shares $\{r_i\}_{i=1}^n$ for a secret $s \in \mathbb{Z}_p$, s.t. $\sum_{i=1}^n r_i \mod p = s$. To sign a message $m_i \in \mathbb{Z}_p$, $\mathcal{DO}_i$ with $\mathbf{sk}_i = (s_i, h_{i,0}^{\beta}, h_{i,1}^{\beta})$ generates a signature $\sigma_i = (\mathcal{W}, \omega_i)$, where $\omega_i = H_1(\mathcal{W})^{r_i} \cdot h_{i,0}^{\beta} \cdot h_{i,1}^{\beta \cdot s_i \cdot m_i}$.
- $\sigma^* \leftarrow \mathsf{AggSig}(\{\sigma_i\}_{i=1}^n, \Omega_I)$: The CSP aggregates n signatures $\{\sigma_i\}_{i=1}^n$ with the same disturbance $\mathcal{W}$ by setting $\sigma^* = (\mathcal{W}, \omega^*)$, where $\omega^* = \prod_{i=1}^n \omega_i$.
- $\{0,1\} \leftarrow \mathsf{VerSig}(\sigma^*, \sum_{i=1}^n m_i, \{\mathbf{pk}_i\}_{i=1}^n, g^s, \Omega_I)$: This algorithm outputs 1 iff Eq. 4 is satisfied:

$$\hat{e}(\omega^*, g) = \hat{e}(g^s, H_1(\mathcal{W})) \cdot \hat{e}(h, \prod_{i=1}^n h_{i,0} \cdot u^{\sum_{i=1}^n m_i}) \tag{4}$$

Since $u = h_{i,1}^{s_i}$ for $i \in [n]$ and $g^{\sum_{i=1}^n r_i} = g^s$, the left-hand side of Eq. 4 expands using the bilinear pairing properties:

$$\begin{aligned}
&\hat{e}(\omega^*, g) = \hat{e}(\textstyle\prod_{i=1}^n H_1(\mathcal{W})^{r_i} \cdot h_{i,0}^{\beta} \cdot h_{i,1}^{\beta \cdot s_i \cdot m_i}, g) \\
&= \hat{e}(H_1(\mathcal{W})^{\sum_{i=1}^n r_i}, g) \cdot \hat{e}(\textstyle\prod_{i=1}^n h_{i,0}^{\beta} \cdot h_{i,1}^{\beta \cdot s_i \cdot m_i}, g) \\
&= \hat{e}(g^s, H_1(\mathcal{W})) \cdot \hat{e}(h, \textstyle\prod_{i=1}^n h_{i,0} \cdot h_{i,1}^{s_i \cdot m_i}) \\
&= \hat{e}(g^s, H_1(\mathcal{W})) \cdot \hat{e}(h, \textstyle\prod_{i=1}^n h_{i,0} \cdot \prod_{i=1}^n u^{m_i}) \\
&= \hat{e}(g^s, H_1(\mathcal{W})) \cdot \hat{e}(h, \textstyle\prod_{i=1}^n h_{i,0} \cdot u^{\sum_{i=1}^n m_i}),
\end{aligned}$$

which is the right-hand side as required, thus validating the correctness of the MSAS scheme. Since g^s that gathering the information of n shares is provided by the user instead of the CSP, Eq. 4 does not hold if the CSP overlooks any data source due to the security of the secret sharing scheme, validating the completeness of data sources.

Algorithm 3 Advanced Construction MARS$^+$

Initialization (by the PKG)

Input: Security parameter λ, data owners' identities $\{\mathcal{ID}_i\}_{i=1}^n$
Output: Parameter Ω, keys $\{\alpha, \beta\}$, key pairs $(\mathbf{pk}_i, \mathbf{sk}_i)_{i=1}^n$, auxiliary messages Ψ, shared keys Φ

```
(Ω_E, α) ← ESA.Setup(λ); (Ω_I, β) ← MSAS.Setup(λ)
Chooses random secrets s ∈ Z_p and κ ∈ {0,1}^λ
for i = 1 to n do
    (pk_i, sk_i) ← MSAS.GenKey(ID_i, β, Ω_I)
Ω ← (Ω_E, Ω_I); Ψ ← (g^s, κ); Φ ← {s, κ}
```

ADS Generation (by $\mathcal{DO}_i$)

Input: Dataset $\mathbb{D}_i$, private key pair $(\mathbf{pk}_i, \mathbf{sk}_i)$, key Φ
Output: The ADS $\mathcal{VGT}_i$ and signature σ_i

```
Execute lines 1-3 of MARS^0.ADS Generation
Construct a VG-tree VGT_i based on Def. 4 and Def. 5
σ_i ← MSAS.GenSig(δ_{i,0}, sk_i, s, Ω_I)
```

Merging (by the CSP)

Input: The ADSs and signatures $(\mathcal{VGT}_i, \sigma_i)_{i=1}^n$
Output: The integrated ADS and signature $(\mathcal{VGT}^*, \sigma^*)$

```
Obtain VGT* by merging {VGT_i}_{i=1}^n according to Def. 6
σ* ← MSAS.AggSig({σ_i}_{i=1}^n, Ω_I)
```

VO Construction (by the CSP)

Input: Query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, dataset $\mathbb{D}^*$, an ADS $\mathcal{VGT}^*$
Output: Final result τ^*, integrated VO $\mathcal{VO}^*$

```
Run Query(VGT*.root, D*, R) to output (V*, MN*, UN*)
(π*, τ*) ← ESA.GenProof(V*, Υ, Ω_E)
VO* ← (MN*, UN*, π*).
```

Verification (by the user)

Input: Query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, VO/signature $(\mathcal{VO}^*, \sigma^*)$, final result τ^*, public keys $\{\mathbf{pk}_i\}_{i=1}^n$, auxiliary messages Ψ
Output: Verification report $\mathcal{VR}$

```
VR ← 0                                ▷ 0 indicates verification fails
Pares VO* as (MN*, UN*, π*)
if MN* and UN* pass validation then
    Reconstruct VGT* and δ_0* with Def. 4-Def. 6
    if MSAS.VerSig(σ*, δ_0*, {pk_i}_{i=1}^n, g^s, Ω_I) then
        acc_{V*} ← ∏_{i=1}^n ∏_{VN_{i,x} ∈ MN*} acc_{V_{i,x}}
        if ESA.VerProof(acc_{V*}, Υ, π*, τ*, Ω_E) then
            VR ← 1
```

The security of the MSAS scheme can be easily derived from that of the IBAS scheme as follows: Let $M_i = (s_i \cdot m_i) \mod p$ denote the cipertext of $\mathcal{DO}_i$'s message m_i. The signature generated by IBAS.GenSig is $\sigma_i = (\mathcal{W}, \omega_i, \varphi_i)$, where $\omega_i = H_1(\mathcal{W})^{r_i} \cdot h_{i,0}^{\beta} \cdot h_{i,1}^{\beta M_i}$ and $\varphi_i = g^{r_i}$. In algorithm IBAS.VerSig, the user verifies whether Eq. 5 is satisfied:

$$\hat{e}(\omega^*, g) = \hat{e}(\varphi^*, H_1(\mathcal{W})) \cdot \hat{e}(h, \prod_{i=1}^n h_{i,0} \cdot \prod_{i=1}^n h_{i,1}^{M_i}) \tag{5}$$

where $\omega^* = \prod_{i=1}^n H_1(\mathcal{W})^{r_i} \cdot h_{i,0}^{\beta} \cdot h_{i,1}^{\beta M_i}$ and $\varphi^* = g^{\sum_{i=1}^n r_i}$. Note that the left-hand side of Eq. 5 is the same as that of Eq. 4, and the right-hand side of Eq. 5 evolves as follows:

$$\begin{aligned}
&\hat{e}(\varphi^*, H_1(\mathcal{W})) \cdot \hat{e}(h, \textstyle\prod_{i=1}^n h_{i,0} \cdot \prod_{i=1}^n h_{i,1}^{M_i}) \\
&= \hat{e}(g^s, H_1(\mathcal{W})) \cdot \hat{e}(h, \textstyle\prod_{i=1}^n h_{i,0} \cdot \prod_{i=1}^n h_{i,0}^{s_i \cdot m_i}) \\
&= \hat{e}(g^s, H_1(\mathcal{W})) \cdot \hat{e}(h, \textstyle\prod_{i=1}^n h_{i,0} \cdot \prod_{i=1}^n u^{m_i}) \\
&= \hat{e}(g^s, H_1(\mathcal{W})) \cdot \hat{e}(h, \textstyle\prod_{i=1}^n h_{i,0} \cdot u^{\sum_{i=1}^n m_i})
\end{aligned}$$

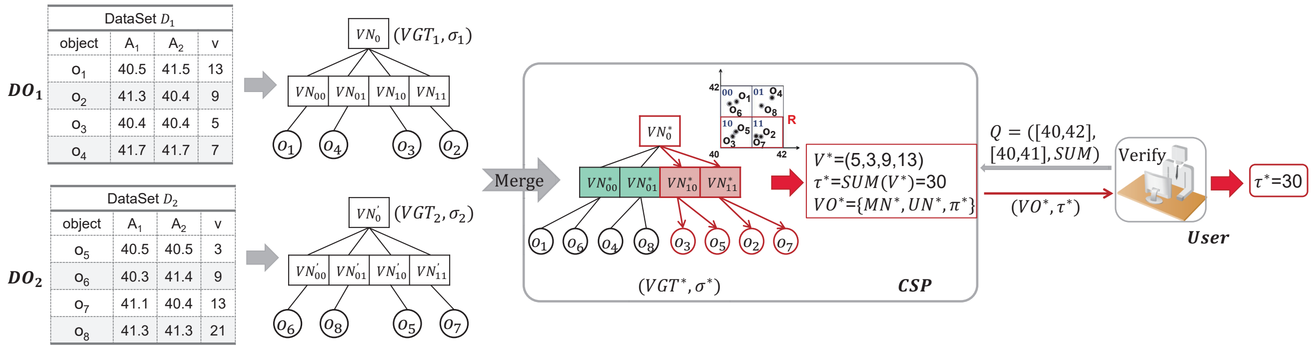


Fig. 5: Exemplary working process of MARS$^+$. A G-tree is constructed under the same requirements of Fig. 4.

which is the same as that of Eq. 4. Therefore, the output of algorithm MSAS.VerSig is equivalent to that of algorithm IBAS.VerSig. The IBAS.VerSig algorithm outputting 1 verifies the authenticity of ciphertexts $\{M_i\}_{i=1}^n$, validating the authenticity of messages $\{m_i\}_{i=1}^n$ and their sum $\sum_{i=1}^n m_i$.

4.3 Details of MARS$^+$

Let $\mathsf{F}_\kappa : \{0,1\}^* \to \mathbb{Z}_p$ be a pseudo-random function (PRF) with key $\kappa \in \{0,1\}^\lambda$, and let MSAS be a MSAS scheme described in Section 4.2. The details of MARS$^+$ are shown in Alg. 3 that takes system parameters Ω as the implicit input.

Initialization. The PKG first runs algorithms ESA.Setup and MSAS.Setup to create system parameters Ω and secret keys $\{\alpha, \beta\}$, and then runs algorithm MSAS.GenKey to generate key pairs $(\mathbf{pk}_i, \mathbf{sk}_i)_{i=1}^n$. Besides, it generates auxiliary messages $\Psi = \{\kappa, g^s\}$ and shared keys $\Phi = \{s, \kappa\}$.

ADS Generation. $\mathcal{DO}_i$ builds a d-dimensional G-tree $\mathcal{GT}_i$ according to Def. 1, and extends $\mathcal{GT}_i$ to a VG-tree $\mathcal{VGT}_i$ by incorporating a digest into each tree node as follows:

Definition 4 (Leaf node digest)**.** *For a leaf node $\mathcal{VN}_{i,x} = \langle \mathcal{N}_{i,x}, \delta_{i,x}\rangle$, where $\mathcal{N}_{i,x} = \langle x, gb_{i,x}, \mathit{OBJ}_{i,x}\rangle$ is a G-tree leaf node, the digest $\delta_{i,x}$ is calculated by Eq. 6:*

$$\delta_{i,x} = \mathsf{F}_\kappa(gb_{i,x} || H(\mathbf{acc}_{V_{i,x}})) \mod p \tag{6}$$

where $V_{i,x}$ is the functional values of objects in $\mathit{OBJ}_{i,x}$, and $\mathbf{acc}_{V_{i,x}}$ is the accumulative value. In a special case, if $\mathit{OBJ}_{i,x} = \emptyset$, $\mathbf{acc}_{V_{i,x}}$ is set to $\perp$ and $\delta_{i,x}$ is set to the identity element of $\mathbb{Z}_p$.

Definition 5 (Non-leaf node digest)**.** *For a non-leaf node $\mathcal{VN}_{i,x} = \langle \mathcal{N}_{i,x}, \delta_{i,x}\rangle$, where $\mathcal{N}_{i,x} = \langle x, gb_{i,x}, (i, x_1), \ldots, (i, x_{2^d})\rangle$ is a G-tree non-leaf node, the digest $\delta_{i,x}$ is calculated by Eq. 7:*

$$\delta_{i,x} = \sum\nolimits_{k=1}^{2^d} \delta_{i,x_k} \mod p \tag{7}$$

After creating the VG-tree, $\mathcal{DO}_i$ signs the root digest $\delta_{i,0}$ and produces a signature σ_i by algorithm MSAS.GenSig.

Merging. Once receiving the ADSs from n data owners, the CSP first merges the VG-trees $\mathcal{VGT}_1, \ldots, \mathcal{VGT}_n$ into an integrated VG-tree $\mathcal{VGT}^*$ by combing the digests as follows:

Definition 6 (The digest of a merged node)**.** *A merged node is defined as $\mathcal{VN}_x^* = (\mathcal{N}_x^*, \delta_x^*)$, where $\mathcal{N}_x^* = \langle x, gb_x, \{\mathit{OBJ}_{i,x}\}_{i=1}^n\rangle$ if $\mathcal{VN}_x^*$ is a leaf node, and $\mathcal{N}_x^* = \langle x, gb_x, x_1, \ldots x_{2^d}\rangle$ otherwise. In any case, the digest δ_x^* is calculated by Eq. 8:*

$$\delta_x^* = \sum\nolimits_{i=1}^{n} \delta_{i,x} \mod p \tag{8}$$

Given the integrated VG-tree, it generates an aggregated signature σ^* for the digest δ_x^* by algorithm MSAS.AggSig.

VO Construction. Given a query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, the CSP executes algorithm Query$(\mathcal{VGT}^*.root, \mathbb{D}^*, \mathcal{R})$ to get $(V^*, \mathrm{MN}^*, \mathrm{UN}^*)$. Then, it runs algorithm ESA.GenProof to output the final result $\tau^* = \Upsilon(V^*)$ and relevant proof π^*. The integrated VO is set to $\mathcal{VO}^* = (\mathrm{MN}^*, \mathrm{UN}^*, \pi^*)$. As shown in Alg. 2 (including the boxed codes), the query process slightly differs from that of MARS^0: When the traversal reaches a leaf node, the CSP puts the functional values of candidates objects in n VG-trees into V^*, while putting the label, the grid, and the accumulative values in n VG-trees into MN^* or UN^* depending on if the leaf node is included in the range query $\mathcal{R}$ or not.

Verification. Given the final result τ^* and the integrated VO/signature pair $(\mathcal{VO}^*, \sigma^*)$, the user performs the following two-stage verification to check if the CSP honestly executes the query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$ on n datasets or not.

Stage 1. The user first examines if MN^* and UN^* conforms to the standard in the same way as MARS^0. If so, reconstructs the integrated VG-tree $\mathcal{VGT}^*$ with MN^* and UN^* and then verifies the root digest δ_0^* using the algorithm MSAS.VerSig. Based on the security of the SSS and IBAS schemes, this algorithm outputting 1 means that all the components in MN^* and UN^* are indeed calculated from the original $\mathbb{D}^*$ and $\mathcal{VGT}^*$, no candidate object is skipped, and no data source is omitted. This validates the integrity of range query $\mathcal{R}$ as well as the completeness of data sources.

Stage 2. The user calculates $\prod_{i=1}^n \prod_{\mathcal{VN}_{i,x} \in \mathrm{MN}^*} \mathbf{acc}_{V_{i,x}}$ to obtain $\mathbf{acc}_{V^*}$, the accumulative value of candidate objects from all data sources, and then runs algorithm ESA.VerProof to authenticate the final result τ^*. Due to the security of ESA, this algorithm outputs 1 only when $\tau^* = \Upsilon(V^*)$, validating the correctness of aggregate query.

4.4 Illustrative Example

As for the application scenario in Section 3.3, Fig. 5 illustrates the working process of MARS$^+$:

ADS Generation. After system initialization, each $\mathcal{DO}_i$ independently creates a VG-tree, $\mathcal{VGT}_i$, as shown in Fig. 5, and then produces a signature σ_i for the root digest $\delta_{i,0}$.

Merging. Given $(\mathbb{D}_i, \mathcal{VGT}_i, \sigma_i)_{i=1,2}$, the CSP merges the VG-trees and aggregates signatures. The integrated VG-tree $\mathcal{VGT}^*$ and the aggregated signature are as shown in Fig. 5.

VO Construction. For the united dataset $\mathbb{D}^*$, the CSP runs algorithm Query on $\mathcal{VGT}^*$ to get $(V^*, \mathrm{MN}^*, \mathrm{UN}^*)$. The

TABLE 2: Performance comparison of MARS^0 and MARS^+

	MARS^0		MARS^+	
	CPU Cost	COMM. Cost	CPU Cost	COMM. Cost
PKG	$O(q \cdot C_\wedge + n \times C_\wedge)$	$O(q+n)$	$O(q \cdot C_\wedge + n \times C_\wedge)$	$O(q+n)$
$\mathcal{DO}_i$	$O(n_i \cdot C_\times + N \cdot C_h)$	$O(N)$	$O(n_i \cdot C_\times + N_1 \cdot (C_h + C_f) + N_2 \cdot C_+)$	$O(N)$
CSP	$O(c \cdot C_\times)$	$O(n \cdot m)$	$O(n \cdot N \cdot C_+ + n \cdot C_\times + c \cdot C_\times + n \cdot C_\times)$	$O(n \cdot m_1)$
User	$O(n \cdot m \cdot C_h + c \cdot C_\times + n \cdot (C_\times + C_b + C_{\times_T}))$	$O(n \cdot m)$	$O(n \cdot m_1 \cdot (C_h + C_f) + m_2 \cdot C_+ + (c+n) \cdot C_\times)$	$O(n \cdot m_1)$

$[q]$ is the universal values; n is the number of data owners; N_1 and N_2 are the number of leaf and non-leaf nodes in each VG-tree, m_1 and m_2 are the number of leaf and non-leaf nodes visited in the query process, respectively; c is the total number of candidate objects; n_i is the number of objects in dataset $\mathbb{D}_i$; $m = m_1 + m_2$ is the total number of visited nodes; $N = N_1 + N_2$ is the total number of nodes in a VG-tree.

search process upon the integrated VG-tree is marked by red thick lines, while the matched and unmatched nodes are filled with red and green, respectively. For dataset $\mathbb{D}^*$, the candidate objects in $\mathcal{R}$ are $\{o_2, o_3, o_5, o_7\}$ with $V^* = \{9, 5, 3, 13\}$. It then runs algorithm $\mathsf{ESA.GenProof}$ to get π^* and $\tau^* = 30$ and sets $\mathcal{VO}^* = \{\mathrm{MN}^*, \mathrm{UN}^*, \pi^*\}$.

Verification. The user reconstructs $\mathcal{VGT}^*$ from $\mathcal{VO}^*$ and verifies the root digest δ_0^* by algorithm $\mathsf{MSAS.VerSig}$. If the validation passes, it calculates the accumulative values $\mathbf{acc}_{V^*}$ of all candidate objects and directly authenticates the final result τ^* by algorithm $\mathsf{ESA.VerProof}$.

5 Analysis

5.1 Performance Analysis

Let C_h, C_f, C_b, C_+, $C_{\times_T}$, $C_\times$ and $C_\wedge$ denote the CPU cost of a hash function, a PRF, a bilinear pairing, an addition operation in $\mathbb{Z}_p$, a multiplication operation in $\mathbb{G}_T$ and a multiplication operation, and a power operation in $\mathbb{G}$, respectively. Given n data owners, each $\mathcal{DO}_i$ is supposed to build a 2^d-ary VG-tree $\mathcal{VGT}_i$ of height η over a dataset $\mathbb{D}_i$ that consists of n_i objects. Therefore, each VG-tree includes $N_1 = 2^{d\cdot(\eta-1)}$ leaf nodes and $N_2 = \sum_{k=1}^{\eta-1} 2^{d\cdot(k-1)}$ non-leaf nodes. Given a query $\mathcal{Q} = (\mathcal{R}, \Upsilon)$, the total number of candidate objects in query $\mathcal{R}$ is denoted by c, and the number of leaf and non-leaf nodes visited in algorithm Query is denoted by m_1 and m_2, respectively. The comparison results of CPU and communication costs are shown in Table 2, where the incoming (resp. outgoing) communication costs are considered for the user (resp. the remaindering entities).

Initialization. In MARS^0, the major costs lie in algorithms $\mathsf{ESA.Setup}$ and $\mathsf{IBAS.Genkey}$, which incur cost $O(q \cdot C_\wedge)$ to output system parameters of size $O(q)$ and cost $O(n \times C_\wedge)$ to generate private keys of size $O(n)$, respectively. MARS^+ incurs similar costs as MARS^0.

ADS Generation. In MARS^0, $\mathcal{DO}_i$ produces accumulative values for all objects in $\mathbb{D}_i$ by algorithm $\mathsf{ESA.GenAcc}$, and calculates hash digest for each node, resulting the total CPU cost $O(n_i \cdot C_\times + (N_1 + N_2) \cdot C_h)$. Once the VG-tree is built, it then produces a signature by algorithm $\mathsf{IBAS.GenSig}$, which incurs a constant cost. Compared with MARS^0, MARS^+ requires $\mathcal{DO}_i$ to calculate hash functions and pseudo-random functions to obtain leaf nodes' digests while calculating 2^d additions in $\mathbb{Z}_p$ for each non-leaf node' digest, resulting the total CPU cost $O(n_i \cdot C_\times + N_1 \cdot (C_h + C_f) + N_2 \cdot C_+)$. Besides, both of them outsource a VG-tree of size $O(N_1 + N_2)$ and a constant-size signature.

Merging. In MARS^+, the CSP performs $n \cdot (N_1 + N_2)$ additions in $\mathbb{Z}_p$ to merge n VG-trees, and runs algorithm $\mathsf{MSAS.AggSig}$ to aggregate n signatures, resulting the total CPU cost $O(n \cdot (N_1 + N_2) \cdot C_+ + n \cdot C_\times)$. After merging, the size of ADSs and signatures is reduced from $O(n \times (N_1 + N_2))$ and $O(n)$ to $O(N_1 + N_2)$ and $O(1)$, respectively.

VO Construction. MARS^0 requires the CSP to run the Query algorithm for n times to outputs n VOs, and run algorithm $\mathsf{ESA.GenProof}$ for n times to generate n proofs $\{\pi_i\}_{i=1}^n$ and n intermediate results $\{\tau_i\}_{i=1}^n$. By contrast, MARS^+ requires the CSP to run the Query algorithm once to output an integrated VO, and run algorithm $\mathsf{ESA.GenProof}$ once to generate a proof π^* and a final result τ^*. Suppose that in both constructions, the CSP locally keeps the accumulative values $\{\mathbf{acc}_{V_{i,x}}\}_{i\in[n], x\in[N_1+N_2]}$. Since the cost of algorithm $\mathsf{ESA.GenProof}$ is related to the number of candidate objects, both constructions incur similar the CPU costs $O(c \cdot C_\times)$. The VO size in MARS^0 and MARS^+ is $O(n \cdot (m_1 + m_2))$ and $O(n \cdot m_1)$, respectively.

Verification. In MARS^0, the user reconstructs n VG-trees while running algorithms $\mathsf{ESA.VerProof}$ and $\mathsf{IBAS.VerSig}$ for n times. In MARS^+, the user reconstructs an integrated VG-tree while running algorithms $\mathsf{ESA.VerProof}$ and $\mathsf{MSAS.VerSig}$ once. The CPU costs in MARS^0 and MARS^+ are $O(n \cdot (m_1 + m_2) \cdot C_h + c \cdot C_\times + n \cdot (C_\times + C_b + C_{\times_T}))$ and $O(n \cdot m_1 \cdot (C_h + C_f) + m_2 \cdot C_+ + (c+n) \cdot C_\times)$, respectively.

5.2 Security Analysis

Theorem 1. *MARS^0 achieves integrity, correctness, and completeness, if hash functions are collision resistant, and the IBAS and ESA schemes are secure.*

Proof. The security can be proven by contradiction:

Case 1: For $\mathbf{acc}_{V_{i,x}} \in \mathcal{VO}_i$, the CSP forges a fake set $\widetilde{V}_{i,x}$ by replacing an element in $V_{i,x}$ with a fake element or an empty element, and replaces $\mathbf{acc}_{V_{i,x}}$ with $\mathbf{acc}_{\widetilde{V}_{i,x}}$. In this case, the CSP forges or skips certain objects, trying to compromise the integrity of range queries. Since the accumulative values are collision resistant, $\mathbf{acc}_{V_{i,x}} \neq \mathbf{acc}_{\widetilde{V}_{i,x}}$ when $V_{i,x} \neq \widetilde{V}_{i,x}$. The root hash is signed with the private key of $\mathcal{DO}_i$. Due to the security of the IBAS scheme, the CSP has to generate the same hash root with $\widetilde{\mathcal{VO}}_i$. A forged accumulative value generating the correct hash root of a VG-tree implies a collision to the security of hash functions.

Case 2: The CSP replaces the intermediate result τ_i with a fake value $\widetilde{\tau}_i$, trying to compromise the correctness of aggregate queries. As proven in Case 1, the accumulative values in MN_i are authentic, validating the authenticity of $\mathbf{acc}_{V_i} = \prod_{\mathcal{VN}_{i,x} \in \mathrm{MN}_i} \mathbf{acc}_{V_{i,x}}$. To pass the verification, the CSP needs to construct a forged $\widetilde{\tau}_i$ and a fake proof

$\widetilde{\pi}_i$ making algorithm ESA.VerProof output 1, leading a contradiction to the security of the ESA scheme.

The user verifies the intermediate result obtained from each VG-tree separately. If result integrity and correctness regarding each VG-tree are proven in Case 1 and Case 2, the completeness of data sources is also confirmed. □

Theorem 2. *MARS$^+$ achieves integrity, correctness, and completeness, if hash functions are collision resistant, PRFs are secure, and the MSAS, ESA, and secret sharing schemes are secure.*

Proof. The security can be proven by contradiction:

Case 1: For $\mathbf{acc}_{V_{i,x}} \in \mathcal{VO}^*$, the CSP forges a fake set $\widetilde{V}_{i,x}$ by replacing an element in $V_{i,x}$ with a fake element or an empty element, and replaces $\mathbf{acc}_{V_{i,x}}$ with $\mathbf{acc}_{\widetilde{V}_{i,x}}$. In this case, the CSP forges or skips certain objects, trying to compromise the integrity of range queries. Since the accumulative values and hash functions are collision resistant, $H(\mathbf{acc}_{V_{i,x}}) \neq H(\mathbf{acc}_{\widetilde{V}_{i,x}})$ when $V_{i,x} \neq \widetilde{V}_{i,x}$. The root digest δ_0^* of the integrated VG-tree is signed with the aggregated signature of n data owners. Due to the security of the MSAS scheme, the CSP has to generate a fake digest $\widetilde{\delta}_0^* = (\delta_0^* - \mathsf{F}_\kappa(gb_x||H(\mathbf{acc}_{V_{i,x}})) + \mathsf{F}_\kappa(gb_x||H(\mathbf{acc}_{\widetilde{V}_{i,x}}))) \mod p$ with $\widetilde{\mathcal{VO}^*}$ s.t. $\widetilde{\delta}_0^* = \delta_0^*$. Since the key κ is protected against the CSP, the case that CSP forges a digest passing verification implies a collision to the security of PRFs.

Case 2: The CSP replaces the final result τ^* with a fake value $\widetilde{\tau}^*$, trying to compromise the correctness of aggregate queries. As proven in Case 1, the accumulative values in MN^* are authentic, validating the authenticity of $\mathbf{acc}_{V^*} = \prod_{i=1}^{n} \prod_{\mathcal{VN}_{i,x} \in \mathrm{MN}^*} \mathbf{acc}_{V_{i,x}}$. To pass the verification, the CSP needs to construct a forged $\widetilde{\tau}^*$ and a fake proof $\widetilde{\pi}^*$ making algorithm ESA.VerProof output 1, leading a contradiction to the security of the ESA scheme.

Case 3: The CSP merges only $n-1$ VG-trees to form $\widetilde{\mathcal{VGT}}^*$, aggregates only $n-1$ signatures to form $\widetilde{\sigma}^* = \prod_{i=1}^{n-1} \sigma_i$, and constructs an incomplete $\widetilde{\mathcal{VO}}^*$ from $\widetilde{\mathcal{VGT}}^*$. In this case, the CSP skips a data owner trying to compromise the completeness of data sources. Since n secret shares and their aggregation are protected against the CSP, a fake digest $\widetilde{\delta}^* = \sum_{i=1}^{n-1} \delta_{i,0}$ calculated from $\widetilde{\mathcal{VO}}^*$ making algorithm MSAS.VerSig output 1, leading a contradiction to the security of the secret sharing scheme. □

6 Discussions

6.1 Extension to Dynamic Updates and Rich Queries

MARS allows a user to efficiently authenticate statistical results of selected multi-source data. However, under practical circumstances, each data owner continuously gathers new data and needs to update the outsourced dataset as required, while the user may perform top-K queries or conditional aggregate queries for a better user query experience. Therefore, we will discuss how to extend MARS to support dynamic updates and rich queries.

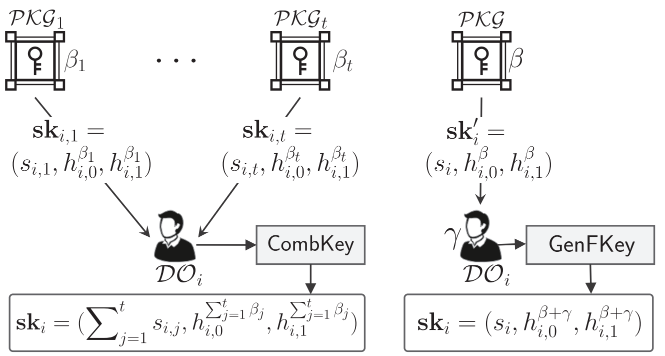
How to Support Updates in Multi-source Environments. The VG-tree structure allows for efficient updates. When a new object is added into a leaf node, the data owner only needs to recalculate relevant accumulative value and recalculate the digests in the path from this leaf node to the root, while resigning the root digest. Unlike MARS^0 where each data owner independently chooses a disturbance and a random number to produce a new signature, MARS$^+$ requires all the data owners to share the disturbance and generate the random numbers using the secret sharing scheme in the signing process. Once the share r_i is updated to r_i' for $i \in [n]$, the data owners need to send the aggregation of new shares $g^{s'} = g^{\sum_{i=1}^{n} r_i'}$ to the user for authenticating queries on the updated data. To reduce the number of interactions, an alternative solution is letting the data owners and the user share a key uk, with which each entity can calculate the new aggregation $g^{s'}$ by itself. Let r_i and $\mathcal{G} = g^s$ denote the current share and aggregation of shares, respectively. A new shared is calculated by $r_i' = r_i + \mathsf{F}_{uk}(\mathcal{ID}_i||\mathcal{W})$ for $i \in [n]$, so that the user can obtain the new aggregation $g^{s'}$ by computing $\mathcal{G}^{\sum_{i=1}^{n} \mathsf{F}_{uk}(\mathcal{ID}_i||\mathcal{W})}$ on his own, in which $\mathcal{W}$ is the disturbance contained in the signature.

How to Enrich Search Functionalities. Besides the normal aggregate operations, e.g., *COUNT*, *SUM*, *MIN*, and *MAX*, MARS can be extended to realize the following types of queries within a given range.

- Top-K queries. To authenticate the top-K functional values within a range $\mathcal{R}$, our solution is letting the CSP prove to the user that the top-i value is the max value for the candidate functional values excluding the top-1, . . . top-$(i-1)$ values. That is, a range-top-K query can be verified by authenticating a range-*MAX* query for K times. Let V^0 denote the candidate objects in $\mathcal{R}$, and let τ_i denote the top-i value in V^0. For $i \in [K]$, the CSP sets $V^i \leftarrow V^{i-1} - \tau_{i-1}$ and runs $\mathsf{GenProof}(V^i, MAX, \Omega_E)$ to output (π_i, τ_i), so that the user can run algorithm $\mathsf{VerProof}(\mathbf{acc}_{V^i}, MAX, \pi_i, \tau_i, \Omega_E)$ to verify if τ_i is the maximal elements in V^i, where $\mathbf{acc}_{V^i}$ can be calculated by $\mathbf{acc}_{V^{i-1}}/g^{s^{\tau_{i-1}}}$, with $\tau_0 = \perp$ and $g^{s^{\tau_0}} = 1$.
- *COUNTIF* and *SUMIF* queries. To support conditional aggregate queries, our main idea is to construct a $(d+1)$-dimensional G-tree from d comparative attributes and a functional attribute, so that the conditional statement can be treated as a query coverage over the $(d+1)$-th attribute. While generating the ADS, a G-tree and a VG-tree are created in a similar way as before, except that the G-tree is a $2^{(d+1)}$-ary tree. Given a query $\mathcal{Q} = (\mathcal{R}, \mathit{COUNTIF})$ or $\mathcal{Q} = (\mathcal{R}, \mathit{SUMIF})$, the CSP first transforms $\mathcal{R}$ to $\mathcal{R}' = \mathcal{R} \wedge R_{d+1}$ where R_{d+1} is the query coverage over the functional attribute corresponding to the conditional statement. Then, the CSP runs algorithm Query to find out the candidate objects within range $\mathcal{R}'$, and performs aggregate operator on these objects. The rest of the process is similar to before.

6.2 Practicability Issues

In our threat model, the PKG as an internal entity is assumed to be fully trusted. This assumption is reasonable for specific applications in which all data sources belong to the same organization (as illustrated in Fig. 1), but it limits the practicality of the proposed scheme. In the real world, the PKG may work abnormally due to external attacks or internal misconfigurations, resulting in potential security problems. In especial, the PKG is responsible for generating signature private keys for data sources, and its malfunction will make our MSAS scheme fail to work. Therefore, we will provide discussions on how to reduce trust in the PKG.



(a) Distributed PKGs (b) Collaborative Key Generation

Fig. 6: The main ideas of extended MSAS schemes.

Distributed PKGs. Inspired by previous work [24], a feasible solution is to distribute the master secret key β among multiple PKGs using threshold cryptography [25] or the secret sharing scheme defined in Section 2.3. In this condition, no single PKG has the complete knowledge of β, and attackers can obtain β only when they break through a sufficient number of PKGs. Suppose that there exist t PKGs, denoted by $\mathcal{PKG}_1, \ldots, \mathcal{PKG}_t$, in the system, and that all PKGs collaborate using the secret sharing scheme. As shown in Fig. 6-(a), our main idea is letting $\mathcal{PKG}_j$ with a secret key share β_j generate a private key share $\mathbf{sk}_{i,j}$ for $\mathcal{DO}_i$, which runs algorithm CombKey to obtain the final private key $\mathbf{sk}_i$. Specifically, the MSAS scheme extended based on distributed PKGs consists of the following algorithms:

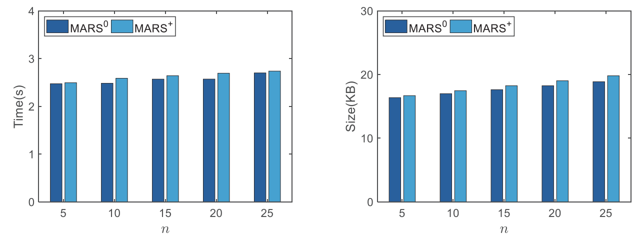
- $(\Omega_I, \{\beta_j\}_{j=1}^t) \leftarrow \mathsf{Setup}(\lambda)$: All PKGs collaboratively run SSS to get secret key shares $\{\beta_j\}_{j=1}^t$ for a master secret key $\beta \in \mathbb{Z}_p$, s.t. $\sum_{j=1}^t \beta_j \mod p = \beta$. The public parameters Ω_I are defined in the same way as those in algorithm MSAS.Setup.
- $(\mathbf{pk}_i, \mathbf{sk}_{i,j}) \leftarrow \mathsf{GenKey}(\mathcal{ID}_i, \beta_j, \Omega_I)$: Before key generation, all PKGs collaboratively choose a random element $s_i \in \mathbb{Z}_p$ and run SSS to get secret key shares $\{s_{i,j}\}_{j=1}^t$ s.t. $\sum_{j=1}^t s_{i,j} \mod p = s_i$. For $\mathcal{DO}_i$ with identity $\mathcal{ID}_i$, $\mathcal{PKG}_j$ calculates the public key $\mathbf{pk}_i = (h_{i,0}, h_{i,1})$ in the same way as algorithm MSAS.GenKey and generates the private key share as $\mathbf{sk}_{i,j} = (s_{i,j}, h_{i,0}^{\beta_j}, h_{i,1}^{\beta_j})$.
- $(\mathbf{sk}_i) \leftarrow \mathsf{CombKey}(\{\mathbf{sk}_{i,j}\}_{j=1}^t)$: On receiving t private key shares $\{\mathbf{sk}_{i,j}\}_{j=1}^t$ from all PKGs, $\mathcal{DO}_i$ calculates $s_i \leftarrow \sum_{j=1}^t s_{i,j}$, $h_{i,0}^{\beta} \leftarrow \prod_{j=1}^t h_{i,0}^{\beta_j}$ and $h_{i,1}^{\beta} \leftarrow \prod_{j=1}^t h_{i,1}^{\beta_j}$, and sets his private key as $\mathbf{sk}_i = (s_i, h_{i,0}^{\beta}, h_{i,1}^{\beta})$.

The remaining algorithms including GenSig, AggSig, and VerSig are constructed in the same way as the original MSAS scheme. In terms of security, external attackers need to breach all the PKGs to gain the master secret key β due to the security of secret sharing scheme. As the value of t increases, the difficulty to compromise all PKGs increases greatly, thereby improving system security.

Collaborative Key Generation. The solution of distributed PKGs has the disadvantage of low efficiency, since each data owner needs to interact with multiple PKGs to obtain an adequate number of private key shares. Inspired by previous work [26], a preferable solution is to enable each data owner $\mathcal{DO}_i$ and a single PKG to collaboratively generate the private key $\mathbf{sk}_i$. In this way, the PKG that participates in part of the key generation process cannot obtain the final private key. As shown in Fig. 6-(b), our main idea is to let the PKG generate an intermediate private key $\mathbf{sk}'_i$ with the master secret key β for $\mathcal{DO}_i$, which takes a random secret γ as the input of algorithm GenFKey to produce the final private key $\mathbf{sk}_i$. Specifically, the main changes in the extension lie in the following aspects:

- $(\Omega_I, \beta) \leftarrow \mathsf{Setup}(\lambda)$: During system initialization, all data owners negotiate a random element $\gamma \in \mathbb{Z}_p$ and sends $g^{\gamma} \in \mathbb{G}$ to the PKG. The PKG randomly chooses a master secret key $\beta \in \mathbb{Z}_p$, and sets the public parameters as $\Omega_I = (\mathbf{pub}, h, H_1, u)$, where $h = g^{\beta+\gamma} \leftarrow g^{\beta} \times g^{\gamma}$ and the remaining components are defined in the same way as those in algorithm MSAS.Setup.
- $(\mathbf{sk}_i) \leftarrow \mathsf{GenFKey}(\mathbf{pk}_i, \mathbf{sk}'_i, \gamma)$: On receiving the intermediate private key $\mathbf{sk}'_i = (s_i, h_{i,0}^{\beta}, h_{i,1}^{\beta})$ from the PKG, $\mathcal{DO}_i$ with public key $\mathbf{pk}_i = (h_{i,0}, h_{i,1})$ and random secret γ calculates $\mu = h_{i,0}^{\beta+\gamma} \leftarrow h_{i,0}^{\beta} \times h_{i,0}^{\gamma}$ and $\nu = h_{i,1}^{\beta+\gamma} \leftarrow h_{i,1}^{\beta} \times h_{i,1}^{\gamma}$. The final private key is set as $\mathbf{sk}_i = (s_i, \mu, \nu)$.

The remaining algorithms including GenKey, GenSig, AggSig, and VerSig are constructed in the same way as those in the original MSAS scheme. In terms of security, the PKG without knowledge of the secret key γ cannot obtain the final private key $\mathbf{sk}_i$. The fact is that no one except the data owner itself can obtain the final private key. Therefore, system security is enhanced as it is hard for attackers to forge signatures even if the PKG is compromised.



(a) Initialization Time (b) Size of Keys and Parameters

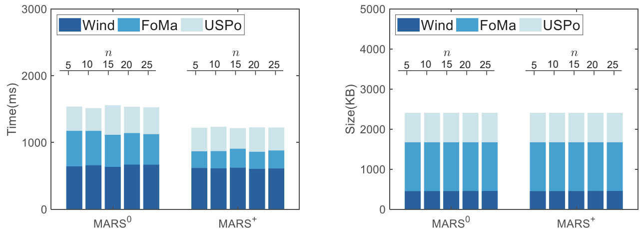
Fig. 7: The initialization cost on the PKG.

7 Evaluation

In this section, we will evaluate the performance of MARS in terms of computation and communication costs. To validate the effectiveness, we conduct experiments on three real datasets. As MARS is the first attempt to authenticate range-aggregate queries on multi-source data, we compare it with PA^2 [15], the verifiable query solution closest to ours.

7.1 Experiment Settings and Datasets

In the experiments, a server with i5 4.4Ghz CPU (6 cores and 12 threads) and a server with Intel Xeon Gold 5218 2.1Ghz CPU (16 cores and 32 threads) are regarded as the PKG and the CSP, respectively; the data owner's program is run on a personal computer equipped with Intel Core i5 3.2GHz CPU and 32GB RAM, and the user-side program is run on a laptop with Intel Core i7 1.8GHz CPU. For the cryptographic algorithms, we set the security parameter λ to 256, and apply SHA-256 and HMAC to implement hash functions and PRFs, respectively. Besides, the experimental



(a) ADS Generation Time (b) ADS Size

Fig. 8: The ADS generation cost on the data owner side.

code is written in Java, and the JPBC library [27] is employed for group operations and bilinear pairing calculations.

We evaluate the experiments on three real datasets, Wind Turbine Scada[5] (Wind for short), FoodMarket[6] (FoMa for short), and US Population By Zip Code[7] (USPo for short). Wind contains $50,525$ records of wind turbines at different times and each record consists of three attributes: wind speed, theoretical power curve, and wind direction. FoMa contains $164,550$ records of shopping transactions and each record consists of three attributes: birthday, membership age, and items. USPo is the largest dataset that contains one million records of nationwide population in the United States and each record consists of three attributes: minimum age, maximum age, and zipcode. In the evaluation, the first two attributes of three datasets are considered comparative attributes, and the third attribute is taken as a functional attribute. Since the parameter size and the computation time of the ESA scheme are determined by the largest possible value q of attributes, we preprocess each dataset by scaling down all attribute values at a specific proportion. Considering n data sources, we divide each dataset into n parts equally and assign each part to a data owner.

According to the performance analysis in Section 5, we know that the number of data owners n and the number of candidate objects c are two important parameters affecting performance. To demonstrate their concrete impacts on the schemes, we set n to $\{5, 10, 15, 20, 25\}$, and the hit rate $r_h = c/|\mathbb{D}^*|$ of range query to $\{1\%, 5\%, 10\%, 20\%\}$. The performance is evaluated with the following metrics: (1) The initialization time; (2) The size of keys and parameters; (3) The ADS generation time; (4) The ADS size; (5) The merging time; (6) The VO construction time; (7) The VO size; (8) The verification time. The first two metrics are tested on the PKG, the 3-th and 4-th metrics are related to data owners, the last metric is relevant to the user, and the remained metrics are tested on the CSP. To minimize deviation, each instance is run at least 100 times to obtain the average value. The experimental results are shown in Fig. 7-13 and Table 3. All the bars in these figures start from the same baseline.

7.2 Experimental Results

Initialization. Fig. 7 illustrates the cost in the initialization phase. From this figure, we can see that the time for generating keys and the size of keys increase as n grows. The reason is obvious, the more the number of data owners, the more the number of keys need to be generated. Moreover, the initialization time and the key size of MARS$^+$ is slightly larger than those of MARS^0. This is because algorithm MSAS.GenKey calculates exponentiation operations to generate an extra key compared with algorithm IBAS.GenKey.

5. https://www.kaggle.com/datasets/berkerisen/wind-turbine-scada-dataset

6. https://recsyswiki.com/wiki/Grocery_shopping_datasets

7. https://www.kaggle.com/datasets/census/us-population-by-zip-code

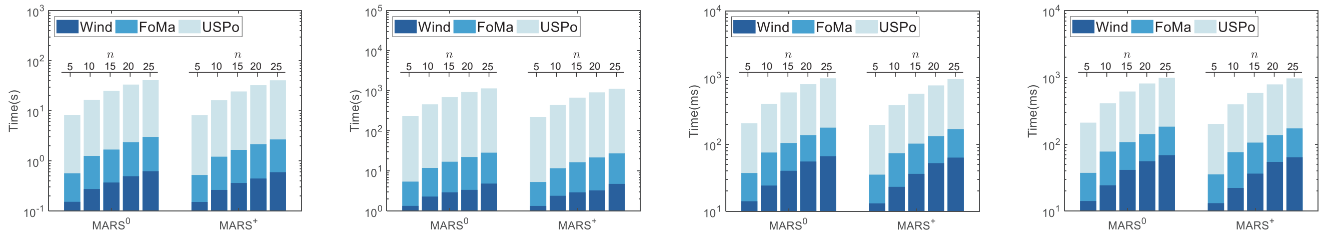
TABLE 3: The merging time on the CSP side (ms)

	$n=5$	$n=10$	$n=15$	$n=20$	$n=25$
MARS$^+$(Wind)	21	35	53	61	65
MARS$^+$(FoMa)	46	76	116	264	292
MARS$^+$(USPo)	50	85	137	283	323

ADS Generation. From Fig. 8, we can observe that n has little influence on the generation time and size of ADSs. The main reason is that each data owner independently generates his own ADS, and this process can be done in parallel among all data owners. Meanwhile, for both constructions, the ADS generation costs on USPo are most expensive, followed by FoMa, and then Wind. This is because the data size of USPo is the largest and that of Wind is the least. In Fig. 8-(a), MARS$^+$ incurs less execution time compared with MARS^0. The main reason is MARS^0 exploits hash operations to calculate the digest of non-leaf nodes yet MARS$^+$ just requires additional operations. In Fig. 8-(b), the ADS size of MARS$^+$ is equal to that of MARS^0 on the same dataset. This is because the VG-trees built by MARS^0 and MARS$^+$ have the same structure with nodes of the same size.

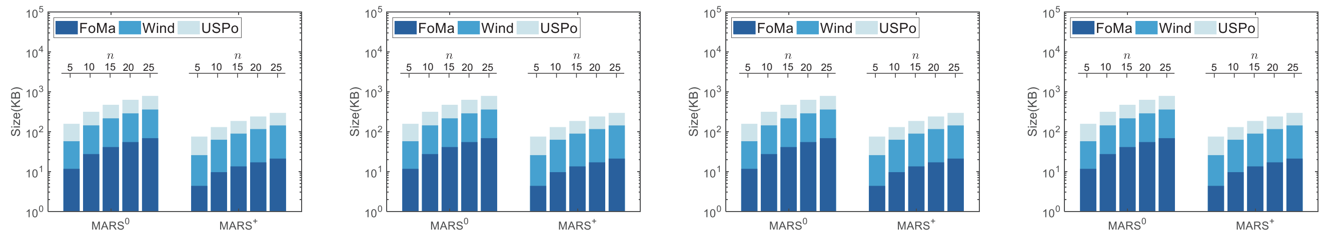
Merging. Since MARS^0 does not merge data, we only evaluate the merging time for MARS$^+$ and demonstrate the results in Table 3. From this table, we can see that the merging time grows linearly along with n increasing. The inherent reason is that the amount of merged data increases as n grows. For the same reason, MARS$^+$ incurs the most and the least merging time on USPo and Wind, respectively, under the same settings. It is noted that the merging time difference between dataset Foma and USPo is very small and the main reason is the VG-trees of both datasets have the same height. In addition, the results also indicate that our merging operation is efficient. For example, the merging time is less than 0.4s for the whole USPo dataset.

VO Construction. To show the impact of parameter n on query performance, we fix the hit rate r_h to 5%. Fig. 9 and Fig. 10 illustrate the VO construction time and VO size for *COUNT*, *SUM*, *MIN*, and *MAX* aggregate queries, respectively. From Fig. 9, we can observe the following trends for MARS^0 and MARS$^+$: (1) The VO construction time of four aggregate queries on three datasets increases as n grows. This is because the bigger n means the more data needs to be processed during VO construction. (2) For the same reason as (1), the VO construction time of four aggregate queries on USPo is the longest, followed by FoMa, and then Wind. (3) Under the same settings, *SUM* query takes the most time to construct VO and performs the worst, while *MAX* and *MIN* queries spend the least time and perform the best. The differences are caused by algorithm ESA.GenProof, which requires the most number of group-related operations for *SUM* query. (4) Compared with MARS^0, MARS$^+$ takes less time to construct VO under the same settings. This is because MARS$^+$ just needs to search an integrated VG-tree



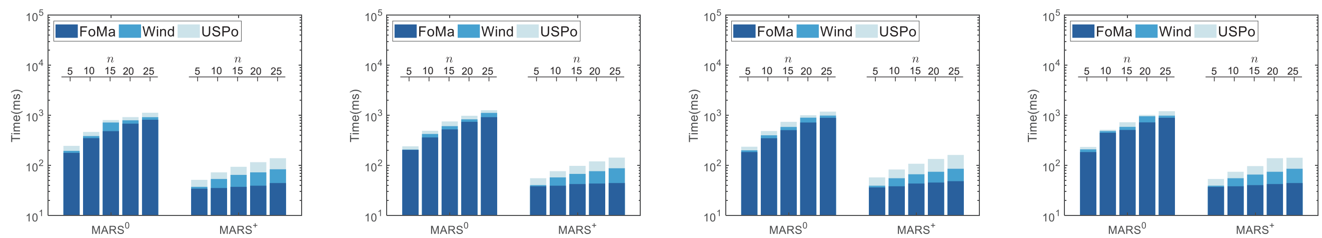
(a) VO Construction Time (COUNT) (b) VO Construction Time (SUM) (c) VO Construction Time (MIN) (d) VO Construction Time (MAX)

Fig. 9: The time of constructing VO on the CSP, where $r_h = 5\%$.



(a) VO Size (COUNT) (b) VO Size (SUM) (c) VO Size (MIN) (d) VO Size (MAX)

Fig. 10: The size of VO transmitted from the CSP to the user, where $r_h = 5\%$.



(a) Verification Time (COUNT) (b) Verification Time (SUM) (c) Verification Time (MIN) (d) Verification Time (MAX)

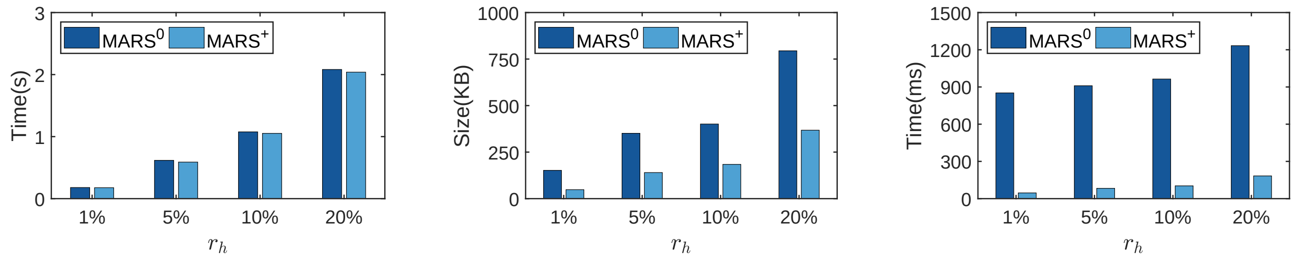
Fig. 11: The verification time on the user side, where $r_h = 5\%$.

yet MARS^0 requires sequentially querying n VG-trees.

From Fig. 10, we can get the following observations: (1) The VO size increases as n grows for both constructions. This is because the total number of visited nodes increases as n grows. (2) The VO size generated by MARS^0 is larger than that of MARS$^+$ under the same settings. The reason is MARS$^+$ just needs to visit one integrated VG-tree hence can reduce some duplicate node information compared with MARS^0. (3) For the same construction, there is almost no difference in VO size among all aggregate queries. This is because the number of visited nodes is mainly impacted by the hit rate r_h of range queries rather than the types of aggregate queries. (4) As for different datasets, the VO size generated from USPo is biggest and that generated from Foma is smallest. This is caused by the distribution of the datasets: FoMa (resp. USPo) forms the most concentrated (resp. decentralized) data distribution, thus incurring the least (resp. most) number of matched and unmatched nodes.

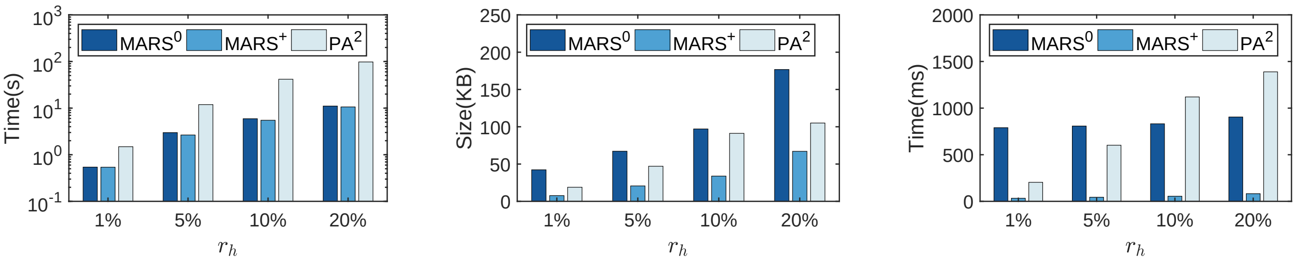
Verification. From Fig. 11, we can observe that: (1) For both constructions, the verification time of all aggregate queries increases with the increase of n. This is because the larger n, the larger VO hence incurring the longer verification time. (2) The verification time of MARS$^+$ is basically an order of magnitude less than that of MARS^0 under the same settings. The main reason is MARS$^+$ just requires reconstructing an integrated VG-tree and running algorithms MSAS.VerSig and ESA.VerProof once. By contrast, MARS^0 reconstructs n VG-trees while running algorithms IBAS.VerSig and ESA.VerProof for n times. (3) Under the same settings, four aggregate queries show only a small difference in execution time. This is because four aggregate queries require different numbers of bilinear pairing operations in algorithm ESA.VerProof. (4) As for different datasets, the verification time on USPo is the longest and that on FoMa is the lowest, and the reason is also caused by the distribution of datasets.

Performance on the Hit Rate. In the experiments, we fix the number of data owners n to 25, and find that the results of different aggregate queries have the same trend as the hit rate r_h changes. Therefore, we choose the most representative *COUNT* query and illustrate the experiment results in Fig. 12. In addition, we compare our work with PA^2 on FoMa dataset and display the results in Fig. 13. Although PA^2 also supports verifiable range-aggregate queries, it essentially differs MARS from the following aspects: (1) It is designed for authenticating queries on single data source.



(a) VO Construction Time (COUNT) (b) VO Size (COUNT) (c) Verification Time (COUNT)

Fig. 12: The cost of VO construction and verification on Wind dataset, where $n = 25$.



(a) VO Construction Time (COUNT) (b) VO Size (COUNT) (c) Verification Time (COUNT)

Fig. 13: Performance comparison with PA^2 on FoMa dataset, where $n = 25$.

Given n data sources, the CSP generates an individual VO for each source and the user authenticates n VOs separately. (2) It supports aggregate operations (*SUM*, *COUNT*, *MIN*, *MAX*) over set-valued data based on bilinear-pairing accumulator [17]. As for a set-valued multiset, the *COUNT* operation plays a similar role as in a numerical set.

From these figures, we can see that the computation and communication costs of all schemes increases as r_h grows. The reason is the larger r_h means that more objects will be retrieved, hence resulting in more time to construct a larger VO and more time to verify the results. In Fig. 13, we also see that: (1) In terms of VO construction, MARS$^+$ incurs the least time to generate the smallest VO, while PA^2 consumes the most time and MARS^0 generates the largest VO. (2) As for verification time, the time difference between MARS$^+$ and PA^2 becomes longer as r_h grows. The performance gain in MARS$^+$ is owing to the adoption of aggregative validation, and the performance penalty in PA^2 is due to the abundant bilinear pairing operations in signature verification. In summary, MARS$^+$ performs best among all schemes in multi-source environments. This is because MARS$^+$ supports aggregative validation, allowing the user to perform one-time verification, while both MARS^0 and PA^2 do not merge data sources, thus requiring the user to authenticate results on each source in sequence.

8 Related Work

8.1 Verifiable Queries in Single Source Environment

As an increasing amount of data is being outsourced to the cloud, a large body of research has been carried out to verify the integrity of query results against an untrusted server. The mainstream approaches normally design an ADS, based on which the server constructs a VO for users to authenticate query results. Merkel hash tree (MHT) [16] and its variants as well as set accumulator [17], [18] are widely used to build ADSs. Zhang et al. [6] proposed the CorrectMR system, which combined Pedersen commitment [28] with Merkle R-tree to authenticate SQL queries. Hu et al. [7] presented the KV-Fresh scheme, in which a linked key span MHT was designed to authenticate freshness of range queries in the key-value store. Wu et al. [8] designed the ServeDB system, in which a SVETree was developed by integrating the hierarchical cube codes into balanced binary tree to verify range queries on encrypted multi-dimensional data. To reduce the computation and storage costs in Ref. [8], Meng et al. [9] designed a verifiable range query scheme VSRQ by combining accumulator technology and G-tree. Meanwhile, they improved the G-tree structure by dividing tree nodes adaptively to accelerate the query process.

In addition, several studies have been conducted to accomplish the verification of rich query expressions. Li et al. [10] designed a verifiable fuzzy query scheme VRFMS by exploiting Bloom filter, locality-sensitive hashing and homomorphic message authentication code (MAC). Our previous work [11] proposed a verifiable top-K search scheme VDERS, which constructed a ranked verifiable matrix to record the ranking information and encoded it with RSA accumulator. Yung et al. [12] presented the VB method based on the Voronoi diagram to authenticate moving kNN query. Cui et al. [13] proposed a SVkNN scheme, which designed a grid-based index on the basis of Voronoi diagram to achieve verifiable kNN query on the encrypted data. Wang et al. [14] presented the DynPilot solution, which designed a DSV-tree to accomplish verifiable location-based skyline query. Xu et al. [15] designed an authentication scheme PA^2 for range-aggregate queries on set-valued data by combining G-tree and bilinear-pairing accumulators. However, the above verification schemes are designed for the single source environment. When being applied to the multi-source scenario, the user has to verify the results obtained from each source separately. That is, these schemes have limited scalability since the verification costs increase as the number of data sources increases.

8.2 Verifiable Queries in Multi-source Environment

With the advent of the big data era, how to provide users with verifiable query results from multi-source fused data has been widely concerned. However, the research in this field is still in its infancy, and there are few verification schemes designed for a multi-source environment. Chen et al. [29] designed a homomorphic secret-sharing seal to aggregate the inputs from multiple sources. Moreover, they proposed two query verification schemes based on G-tree and R-tree to verify the integrity and correctness of multi-dimensional data from multiple sources. Chandrasekhar et al. [30] exploited the multi-trapdoor hashing scheme [31] to aggregate labels of data elements from multiple sources in a verifiable way. Sun et al. [32] proposed ABKS-UR, an authorized keyword search scheme on encryption data, where an authorized user can authenticate search results from multiple data owners. Lu et al. [33] designed EncGD, a count query scheme with the verification functionality for multi-source dynamic DNA data. Tong et al. [34] proposed the VFIRM scheme based on dual secure k-nearest neighbor technique and homomorphic MAC to realize verifiable encrypted image retrieval in multiple data-owners environments. Gupta et al. [35] designed the Obscure scheme to verify aggregated queries with conjunctive or disjunctive predicates based on the secret-sharing technique [20]. However, their work distributed data across multiple servers under the assumption that there were at least $c > 2$ non-communicating servers. In summary, existing verification schemes designed for multi-source environment either require multiple non-colluding servers or support only simple aggregate queries, and thus cannot be applied to verify statistical results of selected data from multiple sources.

9 Conclusion

In this paper, we propose a multi-source authenticated range-aggregate scheme, MARS, to ensure the correctness of statistical results in a cloud-based data fusion environment. Due to the combinableness property of the VG-tree and MSAS scheme, the user can efficiently perform aggregative validation by using an integrated VO and an aggregate signature. To show the benefits of aggregative validation, two constructions are provided and evaluated under different parameters. The formal analyses and empirical studies validate the security and practicality of MARS, respectively. As part of our future work, we will try to combine MARS with privacy-preserving techniques, such as searchable encryption [36], [37] and order-preserving encryption [38] to further ensure the confidentiality of multi-source data.

References

[1] J. Liu, J. Huang, R. Sun, H. Yu, and R. Xiao, "Data fusion for multi-source sensors using GA-PSO-BP neural network," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[2] Y. Dai, J. Liu, X. Ren, and Z. Xu, "Adversarial training based multi-source unsupervised domain adaptation for sentiment analysis," in *Proc. of AAAI*, 2020.

[3] P. Bagga, A. K. Das, M. Wazid, J. J. Rodrigues, K.-K. R. Choo, and Y. Park, "On the design of mutual authentication and key agreement protocol in internet of vehicles-enabled intelligent transportation system," *IEEE Transactions on Vehicular Technology*, 2021.

[4] L. Xu, S. Sun, X. Yuan, J. K. Liu, C. Zuo, and C. Xu, "Enabling authorized encrypted search for multi-authority medical databases," *IEEE Transactions on Emerging Topics in Computing*, 2021.

[5] Amazon cloud directory. [Online]. Available: https://aws.amazon.com/cloud-directory/

[6] B. Zhang, B. Dong, and W. H. Wang, "CorrectMR: Authentication of distributed sql execution on mapreduce," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[7] Y. Hu, X. Yao, R. Zhang, and Y. Zhang, "Freshness authentication for outsourced multi-version key-value stores," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[8] S. Wu, Q. Li, G. Li, D. Yuan, X. Yuan, and C. Wang, "Servedb: Secure, verifiable, and efficient range queries on outsourced database," in *Proc. of ICDE*, 2019.

[9] Q. Meng, J. Weng, Y. Miao, K. Chen, Z. Shen, F. Wang, and Z. Li, "Verifiable spatial range query over encrypted cloud data in vanet," *IEEE Transactions on Vehicular Technology*, 2021.

[10] X. Li, Q. Tong, J. Zhao, Y. Miao, S. Ma, J. Weng, J. Ma, and K.-K. R. Choo, "VRFMS: Verifiable ranked fuzzy multi-keyword search over encrypted data," *IEEE Transactions on Services Computing*, 2023.

[11] Q. Liu, Y. Tian, J. Wu, T. Peng, and G. Wang, "Enabling verifiable and dynamic ranked search over outsourced data," *IEEE Transactions on Services Computing*, 2022.

[12] D. Yung, Y. Li, E. Lo, and M. L. Yiu, "Efficient Authentication of continuously moving k NN queries," *IEEE Transactions on Mobile Computing*, 2015.

[13] N. Cui, X. Yang, B. Wang, J. Li, and G. Wang, "SVkNN: Efficient secure and verifiable k-nearest neighbor query on the cloud platform," in *Proc. of ICDE*, 2020.

[14] Z. Wang, L. Zhang, X. Ding, K. -K. R. Choo, and H. Jin, "A dynamic-efficient structure for secure and verifiable location-based skyline queries," *IEEE Transactions on Information Forensics and Security*, 2023.

[15] C. Xu, Q. Chen, H. Hu, J. Xu, and X. Hei, "Authenticating aggregate queries over set-valued data with confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[16] Y. Sun, Q. Liu, X. Chen, and X. Du, "An adaptive authenticated data structure with privacy-preserving for big data stream in cloud," *IEEE Transactions on Information Forensics and Security*, 2020.

[17] L. Nguyen, "Accumulators from bilinear pairings and applications," in *Proc. of CT-RSA*, 2005.

[18] Y. Zhang, J. Katz, and C. Papamanthou, "An expressive (zero-knowledge) set accumulator," in *Proc. of EuroS&P*, 2017.

[19] D. Boneh and X. Boyen, "Short signatures without random oracles and the SDH assumption in bilinear groups," *Journal of Cryptology*, 2008.

[20] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979.

[21] C. Gentry, and Z. Ramzan, "Identity-based aggregate signatures," in *Proc. of PKC*, 2006.

[22] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil pairing," in *Proc. of ASIACRYPT*, 2001.

[23] E. S. Schwartz and B. Kallick, "Generating a canonical prefix encoding," *Communications of the ACM*, 1964.

[24] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *Proc. of CRYPTO*, 2001.

[25] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *Proc. of EUROCRYPT*, 1999.

[26] V. Goyal, "Reducing trust in the PKG in identity based cryptosystems," in *Proc. of CRYPTO*, 2007.

[27] A. D. Caro, and V. Iovino, "jPBC: Java pairing based cryptography," in *Proc. of ISCC*, 2011.

[28] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *Proc. of EUROCRYPT*, 1991.

[29] Q. Chen, H. Hu, and J. Xu, "Authenticated online data integration services," in *Proc. of SIGMOD*, 2015.

[30] S. Chandrasekhar, and M. Singhal, "Efficient and scalable query authentication for cloud-based storage systems with multiple data sources," *IEEE Transactions on Services Computing*, 2017.
[31] S. Chandrasekhar, and M. Singhal, "Multi-trapdoor hash functions and their applications in network security," in *Proc. of CNS*, 2014.
[32] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, 2016.
[33] D. Lu, M. Li, Y. Liao, G. Tao, and H. Cai, "Verifiable privacy-preserving queries on multi-source dynamic DNA datasets," *IEEE Transactions on Cloud Computing*, 2022.
[34] Q. Tong, Y. Miao, L. Chen, J. Weng, X. Liu, K. K. R. Choo, and R. H. Deng, "VFIRM: Verifiable fine-grained encrypted image retrieval in multi-owner multi-user settings," *IEEE Transactions on Services Computing*, 2022.
[35] P. Gupta, Y. Li, S. Mehrotra, N. Panwar, S. Sharma, and S. Almanee, "Obscure: Information-theoretically secure, oblivious, and verifiable aggregation queries on secret-shared outsourced data," *IEEE Transactions on Knowledge and Data Engineering*, 2022.
[36] S.-F. Sun, C. Zuo, J. K. Liu, A. Sakzad, R. Steinfeld, T. H. Yuen, X. Yuan, and D. Gu, "Non-interactive multi-client searchable encryption: Realization and implementation," *IEEE Transactions on Dependable and Secure Computing*, 2022.
[37] Q. Liu, Y. Peng, H. Jiang, J. Wu, T. Wang, T. Peng, and G.Wang, "SlimBox: Lightweight packet inspection over encrypted traffic," *IEEE Transactions on Dependable and Secure Computing*, 2022.
[38] X. Li, T. Xiang, S. Guo, H. Li, and Y. Mu, "Privacy-preserving reverse nearest neighbor query over encrypted spatial data," *IEEE Transactions on Services Computing*, 2022.

Qin Liu received her B.Sc. in Computer Science in 2004 from Hunan Normal University, China, received her M.Sc. in Computer Science in 2007, and received her Ph.D. in Computer Science in 2012 from Central South University, China. She has been a Visiting Student at Temple University, USA. Her research interests include security and privacy issues in cloud computing. Now, she is an Associate Professor in the College of Computer Science and Electronic Engineering at Hunan University, China.

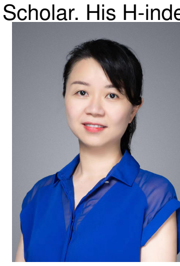
Yu Peng is currently working toward the PhD degree with the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include the security and privacy issues in cloud computing, networked applications and blockchain.

Qian Xu received her B.Sc. in Network Engineering in 2020 from Xiangtan University, China. Currently, she is studying for a Master's degree in the School of Computer Science and Electronic Engineering at Hunan University, China. Her research interests include security and privacy issues in cloud computing.

Hongbo Jiang received the PhD degree from Case Western Reserve University, in 2008. After that, he joined the faculty of the Huazhong University of Science and Technology as a full professor and the dean of the Department of Communication Engineering. Now, he is a full professor with the College of Computer Science and Electronic Engineering, Hunan University. His research concerns computer networking, especially algorithms and protocols for wireless and mobile networks.He is serving as an editor for the IEEE/ACM Transactions on Networking, associate editor for the IEEE Transactions on Mobile Computing, and associate technical editor for the IEEE Communications Magazine.

Jie Wu is the Chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University, Philadelphia, PA, USA. Prior to joining Temple University, he was a Program Director at the National Science Foundation and a Distinguished Professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu has regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE TRANSACTIONS ON SERVICE COMPUTING, and Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Tian Wang received his BSc and MSc degrees in Computer Science from the Central South University in 2004 and 2007, respectively. He received his PhD degree in City University of Hong Kong in Computer Science in 2011. Currently, he is a professor in the Institute of Artificial Intelligence and Future Networks, Beijing Normal University & UIC. His research interests include internet of things, edge computing and mobile computing. He has 27 patents and has published more than 200 papers in high-level journals and conferences. He has more than 11000 citations, according to Google Scholar. His H-index is 59.



Tao Peng received the B.Sc. in Computer Science from Xiangtan University, China, in 2004, the M.Sc. in Circuits and Systems from Hunan Normal University, China, in 2007, and the Ph.D. in Computer Science from Central South University, China, in 2017. Now, she is an Associate Professor of School of Computer Science and Cyber Engineering, Guangzhou University, China. Her research interests include network and information security issues.

Guojun Wang received B.Sc. degree in Geophysics, M.Sc. degree in Computer Science, and Ph.D. degree in Computer Science, at Central South University, China, in 1992, 1996, 2002, respectively. He is a Pearl River Scholarship Distinguished Professor of Higher Education in Guangdong Province, a Doctoral Supervisor of School of Computer Science and Cyber Engineering, Guangzhou University, China, and the Director of Institute of Computer Networks at Guangzhou University. He has been listed in Chinese Most Cited Researchers (Computer Science) by Elsevier in the past eight consecutive years (2014-2021). His research interests include artificial intelligence, big data, cloud computing, Internet of Things (IoT), blockchain, trustworthy/dependable computing, network security, privacy preserving, recommendation systems, and smart cities. He is a Distinguished Member of CCF, a Member of IEEE, ACM and IEICE.

Shaobo Zhang received the B.Sc. and M.Sc. degree in computer science both from Hunan University of Science and Technology, Xiangtan, China, in 2003 and 2009 respectively, and the Ph.D. degree in computer science from Central South University, Changsha, China, in 2017. He was a Post Doctoral Researcher of software engineering with the National University of Defense Technology, Changsha, China, from 2018 to 2022. He has published more than 60 referred papers in his research interests, which include data privacy, information security, cloud computing etc. He is currently an associate professor at School of Computer Science and Engineering of the Hunan University of Science and Technology, China. His research interests include privacy and security issues in social networks and cloud computing.