MPV: Enabling Fine-Grained Query Authentication in Hybrid-Storage Blockchain

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

Abstract—Due to the large-scale data streams produced by distributed terminals, hybrid-storage blockchain (HSB) that combines on-chain and off-chain storages has emerged as a promising solution for secure data storage in decentralized applications. Because all the raw data is outsourced to an untrusted service provider (SP), existing solutions suggest to utilize an on-chain authenticated data structure (ADS) to verify query results retrieved off-chain. However, existing solutions support only *coarse-grained authentication* making a user abandon all the query results once the validation fails. In this paper, we focus on realizing *fine-grained authentication* for range queries, enabling a user to distinguish authentic data from falsified results. Considering the heavy gas consumption of on-chain storage, we propose two multi-dimensional parity-based verification (MPV) schemes with a trade-off between off-chain and on-chain efficiencies. Our main idea is to design an accumulator-based ADS to summarize well-designed verifiable hypercubes, so that fake results can be quickly located by combining multi-dimensional faces failed validation. Compared with previous solutions, our MPV schemes allow a user to make efficient use of query results by filtering out errors, and thus have higher data utility. The detailed security analysis and extensive experiments demonstrate the security and effectiveness of our MPV schemes, respectively.

Index Terms—Blockchain, hybrid storage, fine-grained authentication, range query, error locating.

1 Introduction

A blockchain is a distributed append-only ledger built upon a chain of blocks maintained by a peer-to-peer network [1]. Owing to the feature of immutability and tamper resistance, blockchain is emerging as a promising solution to secure data storage in decentralized applications, such as internet of things (IoT) and crowdsourcing [2]. However, as the ever-increasing amount of data produced by terminal devices (e.g., IoT devices), storing all the raw data directly on the blockchain lacks scalability and efficiency, enabling the hybrid-storage blockchain (HSB) that combines on-chain and off-chain storages to become a preferred alternative [3].

A typical HSB architecture is shown in Fig. 1, where a data owner outsources massive key-value data streams to an off-chain service provider (SP) [4], while storing only the metadata (e.g. cryptographic hashes) on the blockchain. Since the SP is not fully trusted, existing solutions put forward to incorporate an authenticated data structure (ADS) into the on-chain metadata, so that a user can authenticate the query results retrieved off-chain by validating an verification object (VO) [5] constructed by the SP. Although query credibility is ensured, previous solutions support only

Qin Liu, Yu Peng, Mingzuo 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, mzxu}@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

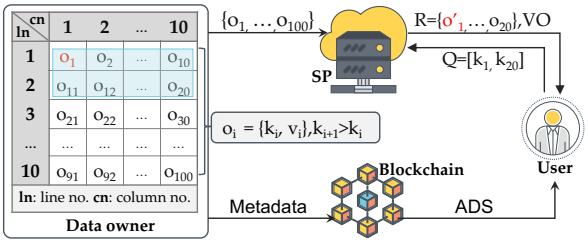


Fig. 1: HSB architecture.

coarse-grained authentication, in which a user accepts or abandons all the query results depending on if the validation passes or not. In practice, the most common queries (e.g., range queries) would retrieve a lot of matched objects, only a small part of which are tampered, making most of the results still valid when the query validation fails. Therefore, *fine-grained authentication* that allows a user to distinguish authentic data from falsified results is essential to improve the data utility in an HSB environment.

As illustrated in [6], range queries are most frequently used in blockchain applications. For a better query experience, this paper is devoted to achieving fine-grained authentication of range queries over a key-value HSB. We propose two multi-dimensional parity-based verification (MPV) schemes with support for error locating. Our main idea is to construct an n-dimensional verifiable hypercube $\mathbf{Q}_s$ for each key k_s, while generating an accumulator-based ADS to summarize all the faces of hypercube $\mathbf{Q}_s$ so that each fake result can be located by n faces failed validation. In the special case of $n = 2$, the hypercube $\mathbf{Q}_s$ is degenerated into a matrix $\mathbf{M}_s$, and the error-locating process is similar to the two-dimensional parity-based method. Taking the example shown in the Fig. 1, where results $R = \{o_1, \ldots, o_{20}\}$ of query $\mathcal{Q} = [k_1, k_{20}]$ spread over the first two lines of matrix

$\mathbf{M}_s$. If the real object o_1 is replaced by a fake object o_1', the verification regarding the 1-st line and the 1-st column of matrix $\mathbf{M}_s$ fails. By combining the line and column numbers failed validation, the user can locate the fake object and make use of the remaining 95% of results.

The basic MPV scheme helps to improve the utility of off-chain data, but renders the size of on-chain ADS to grow as the increase of data scale. As the operation of storing data to the blockchain consumes expensive gas cost [7], we trade higher off-chain computational costs for lower on-chain storage costs, and further propose a gas-efficient MPV scheme to generate a constant-size ADS. Detailed discussions are also provided to support dynamic updates and freshness authentication (i.e., ensuring that the object returned is the latest version for the searching key). The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work on fine-grained query authentication with support for error locating in the key-value HSB environment.
- We propose two MPV schemes with a trade-off between off-chain and on-chain efficiencies. Compared with previous solutions, the main advantage of this work is *high data utility*, i.e., the user can make efficient use of query results by filtering out errors.
- We conduct formal security analyses and an empirical study to validate the proposed MPV schemes.

The remainder of the paper is organized as follows. We introduce the related work in Section 2 before formulating the problem in Section 3. We construct the proposed schemes in Section 4 and Section 5 before analyzing the security in Section 6. After providing discussions in Section 7, we evaluate the performance in Section 8. Finally, we conclude this paper in Section 9.

2 Related Work

2.1 Verifiable Query and Error Locating

To authenticate query results returned by an untrusted party, existing work mainly constructs an ADS based on verifiable technologies, such as digital signature [8], Merkle hash tree (MHT) [9], and accumulator [10]. Xu et al. [11] designed a vector neighbor chain, which concatenated a data item with its left neighbor and right neighbor to authenticate the integrity of range query results. Wu et al. [12] proposed a verifiable range query system ServeDB, which built a SVETree by incorporating the hierarchical cubes into MHTs to realize results validation. To reduce the computation and storage costs in work [12], Meng et al. [13] designed a verifiable spatial range query scheme VSRQ by combining accumulator technology and G-tree. To enrich the query conditions, Zhang et al. [14] devised a verifiable SQL query scheme CorrectMR by integrating the Pedersen commitment into Merkle R-tree. Xu et al. [15] proposed a verifiable aggregate query scheme PA^2 by combining G-tree and bilinear-pairing accumulators. Gupta et al. [16] designed a Obscure scheme, which realized verifiable aggregate queries with complex predicates based on the secret-sharing technique.

As for freshness queries, Jin et al. [17] realized instant freshness check for the retrieved data by combining broadcast encryption, key regression, and MHT. Zhu et al. [18] proposed a GSSE scheme that guaranteed data freshness by developing a timestamp-chain. Hu et al. [19] proposed a freshness authentication scheme KV-Fresh, which designed a linked key span MHT embedded with time relationship between records to provide real-time freshness guarantee. As for error locating, Mu et al. [20] accomplished range query verification and result correction through signature chain and Reed Solomon code. Kittur et al. [21] proposed an error locating scheme based on cyclic redundancy check. Li et al. [22] detected illegal RSA signatures by randomly filling each signature into a matrix and verifying each row and each column. Xu et al. [23] proposed the CUBE scheme, which used cube segmentation to locate erroneous outsourced data. Yuan et al. [24] designed a reverse signature aggregation tree to support batch verification and error locating in cloud file auditing. On the whole, most of the above schemes focus on query verification other than error locating. Existing error locating solutions are mainly designed for achieving proofs of retrievability and provable data possession, and thus cannot support fine-grained query authentication services.

TABLE 1: Comparison of Verifiable Query Schemes

Ref.	Verification	Error Locating	Freshness	Blockchain/HSB
[12]	✓	×	×	×
[15]	✓	×	×	×
[23]	✓	✓	×	×
[25]	✓	×	×	✓
[32]	✓	×	×	✓
Ours	✓	✓	✓	✓

2.2 Verifiable Query in Blockchain

Xu et al. [6] proposed vChain, which used accumulator-based ADSs to verify the boolean range queries and subscription queries in blockchain. To improve query efficiency, their follow-up work [25] devised vChain+ based on the slid window index. Dai et al. [26] proposed a LVQ scheme to realize verifiable historical transactions in Bitcoin systems by integrating Bloom filter (BF) into MHT. Peng et al. [27] proposed a FalconDB system to realize range query verification in blockchain by combining accumulator and MHT. However, the above verifiable schemes upload the raw data together with ADSs to the blockchain. With the explosion of data volume, it will cause huge storage burden and frequent write operations on blockchain, resulting in a performance bottleneck. To improve the scalability of blockchain systems, a common practice is combining the blockchain with an off-chain storage to form an HSB system, so that only the small-sized ADSs are stored on the blockchain, while the enormous raw data is outsourced.

To ensure the off-chain SP faithfully executes queries, Rahman et al. [28] designed a verification framework for industrial internet of things by exploiting Guillou-Quisquater multisignature. Zhu et al. [29] devised SEBDB to provide verifiable SQL queries based on Merkle B-tree. Pei et al. [30] proposed a verifiable semantic query scheme over HSB by exploiting Merkle semantic Trie-based indexing technique. Wu et al. [31] designed a verifiable query layer VQL deployed in the cloud and used Merkle Patricia Tree to construct ADSs so as to provide verifiable query services for blockchain systems. To reduce the gas consumption of smart contracts, Zhang et al. [7] proposed a gas efficiency structure GEM^2-tree by replacing the expensive write operation with lightweight operations (such as reading and computing op-

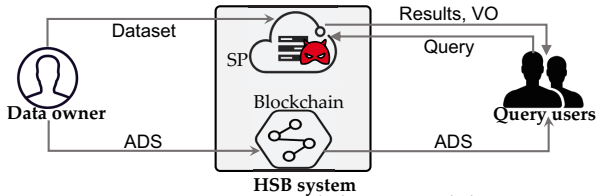


Fig. 2: System and threat models.

erations). Their follow-up work [32] provided an optimized structure based on chameleon vector commitment and BF. Zhang et al. [33] proposed a blockchain-based auditing scheme to protect data integrity for a multi-cloud storage. Once the verification failed, the smart contract asked an organizer to find the malicious SPs thereby accomplishing faults locating. However, their scheme required an organizer to manage the interaction between users and SPs. In summary, existing verification schemes in the HSB system mainly focus on improving query efficiency and enriching query expressions without considering the problems of freshness verification and error locating. The comparison between our work and previous work is shown in Table 1.

3 Problem Formulation

3.1 System and Threat Models

As illustrated in Fig. 2, our system model consists of a data owner, a HSB system, and multiple query users, where the HSB system is composed of a blockchain with smart contract functionality and an off-chain service provider (SP).

• **Data owner.** The data owner is responsible for collecting data and constructing the dataset $\mathcal{D}$, which is modeled as a multi-version key-value store consisting of a collection of key/object-set pairs. Each object set associated with an unique key k contains multiple object versions, each represented as an update value v, and a timestamp t indicating when the update happens. For cost effectiveness, the data owner outsources the dataset $\mathcal{D}$ to the SP once adequate data has been collected. To enable verifiable queries, the data owner also constructs a small-sized ADS and uploads it to the blockchain. Furthermore, the data owner is allowed to update the on-chain ADS once the new collected data is pushed to the off-chain key-value store.

• **Service provider.** The SP, which centralizes abundant computation and storage resources, is responsible for providing data storage and query services in a pay-as-you-use manner. Compared with the blockchain, the SP can deliver storage and query services in a cheaper and more scalable way. Therefore, the data owner outsources the whole dataset $\mathcal{D}$ to the SP but uploads only small-sized ADSs to the blockchain. However, the SP is not fully trusted and thus needs to prove to the data owner and query users that the offered services are credible. On receiving the query $\mathcal{Q}$, the SP performs calculations on $\mathcal{D}$ to obtain results R while constructing a verification object $\mathcal{VO}$ accordingly and finally returns $(R, \mathcal{VO})$ to the query user.

• **Blockchain.** The blockchain is a decentralized database which consists of multiple blocks, and is maintained by all nodes in the network. Due to the immutability property, the blockchain is used to store ADSs. On getting the ADS from the data owner, the blockchain executes smart contract to write the ADS in a new block. During query authentication, the query user submits a query transaction to the blockchain, which executes smart contract and returns the corresponding ADS.

TABLE 2: Summary of Notations

Notations	Descriptions
OBJ_s	The object set of key k_s
d_s	The size of object set $\lvert\mathsf{OBJ}_s\rvert$
n, N	The number of dimensions/keys
RO_s	The ranked object set $\{(k_s, v_c, t_c, c)\}_{c\in[d_s]}$ for key k_s
$\mathbf{M}_s$, $\mathbf{Q}_s$	The verifiable matrix/hypercube of key k_s
μ_s, ν_s	The size of matrix $\mathbf{M}_s$ or hypercube $\mathbf{Q}_s$
$\mathbf{M}_s[i][*]$	The set of elements in the i-th line of matrix $\mathbf{M}_s$
$\mathbf{M}_s[*][j]$	The set of elements in the j-th column of matrix $\mathbf{M}_s$
$\mathcal{F}_{i,j}$	The j-th face at the i-th dimension
$\mathbf{Q}_s[i,j]$	The set of elements in face $\mathcal{F}_{i,j}$ of hypercube $\mathbf{Q}_s$
$\delta_{\mathcal{L}_i}$, $\delta_{\mathcal{C}_j}$	The digest of $\mathbf{M}_s[i][*]$ or $\mathbf{M}_s[*][j]$
$\pi_{\mathcal{L}_i}$, $\pi_{\mathcal{C}_j}$	The proof for the subset of $\mathbf{M}_s[i][*]$ or $\mathbf{M}_s[*][j]$
$\delta_{\mathcal{F}_{i,j}}$, $\pi_{\mathcal{F}_{i,j}}$	The digest of $\mathbf{Q}_s[i,j]$ or the proof for $\mathbf{Q}_s[i,j]$'s subset
$\mathbf{acc}(X)$	The accumulative value of set X

• **Query user.** The query user issues a range query $\mathcal{Q} = (k, [t_l, t_u])$ to the SP to retrieve the objects updated within time period $[t_l, t_u]$ regarding the searching key k. On getting the query results and the VO from the off-chain SP, the user authenticates results and locates errors by combining the on-chain ADS and the off-chain VO.

In our threat model, we assume that the data owner, blockchain and query users are honest, but the SP is malicious. That is, the SP may return tampered results unintentionally or intentionally. To authenticate the outsourced query process, the user checks if the query results R satisfy the following conditions:

• **Soundness**. All the objects in results R satisfy the query condition, and haven't been tampered with.

• **Completeness**. No object satisfying the query condition is skipped in the search process.

3.2 Notations

Let $\lambda \in \mathbb{N}$ be the security parameter throughout this paper. Let $\{0,1\}^n$ be the set of binary strings of length n and $\{0,1\}^*$ the set of binary strings of finite length. Notation $[x, y]$ represents the set of integers $\{x, \ldots, y\}$ and can be abbreviated as $[y]$ when $x = 1$. For a finite set X, $|X|$ denotes its cardinality. Notation $\|$ denotes string concatenation.

The outsourced dataset $\mathcal{D}$ consists of a collection of key/object-set pairs $\{(k_s, \mathsf{OBJ}_s)\}_{s=1}^{N}$. The universal key set is denoted by $\mathsf{KEY} = \{k_1, \ldots, k_N\}$. For each key k_s, the set of multi-version objects is denoted by $\mathsf{OBJ}_s = \{(v_i, t_i)\}_{i=1}^{d_s}$ where $d_s = |\mathsf{OBJ}_s|$. To generate an on-chain ADS, the data owner constructs an n-dimensional hypercube $\mathbf{Q}_s$ and calculates a digest for each face of $\mathbf{Q}_s$. As for VO construction, the SP generates proofs for the set of faces of $\mathbf{Q}_s$ covered by the query results R. When $n = 2$, the hypercube $\mathbf{Q}_s$ is degenerated into a matrix $\mathbf{M}_s$, and the digests and proofs are calculated regarding matrix lines and columns. For quick reference, the most relevant notations are shown in Table 2.

3.3 RSA Accumulator

The ADS and VO are calculated based on RSA accumulator [34] that provides a constant-size digest for a large set and a succinct proof for the (non-)membership test. Let $\mathbf{N} = p \cdot q$ and $\varphi(\mathbf{N}) = (p-1)(q-1)$, where p, q are two large primes. Let g be the generator of a cyclic group $\mathrm{QR}_{\mathbf{N}}$, and let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a collision-resistant hash function. Given public keys $pk = \{\mathbf{N}, (g, \mathrm{QR}_{\mathbf{N}})\}$ and secret keys $sk = \{\varphi(\mathbf{N})\}$, RSA accumulator consists of the following algorithms taking pk as their implicit input.

• $\mathsf{GenAcc}(X) \rightarrow \mathbf{acc}(X)$: Given a set $X = \{x_1, \ldots, x_n\}$ with $x_i \in \{0,1\}^*$, this algorithm generates the accumulative value as $\mathbf{acc}(X) \leftarrow g^{\prod_{i=1}^{n} \mathcal{P}(\mathsf{H}(x_i))} \mod \mathbf{N}$ [1].

• $\mathsf{GenProof}(Y, X) \rightarrow \pi$: This algorithm calculates the proof for a subset Y as $\pi \leftarrow g^{\prod_{x_i \in (X-Y)} \mathcal{P}(\mathsf{H}(x_i))} \mod \mathbf{N}$.

• $\mathsf{VeriWit}(Y, \pi, \mathbf{acc}(X)) \rightarrow \{0,1\}$: This algorithm checks the proof regarding $Y \subseteq X$, and outputs 1 only when $\pi^{\prod_{x_i \in Y} \mathcal{P}(\mathsf{H}(x_i))} \mod \mathbf{N} = \mathbf{acc}(X)$. Based on the collision resistance of hash functions as well as strong RSA assumption, it is difficult for the adversary to find a set $Y' \not\subseteq X$ and a proof π' s.t. $\pi'^{\prod_{x_i \in Y'} \mathcal{P}(\mathsf{H}(x_i))} = \mathbf{acc}(X)$.

4 The Basic MPV Scheme

In this section, we first introduce a strawman construction, denoted by MPV^0, where verifiable matrices are designed for query authentication and error locating, and then we propose an improved construction, denoted by MPV^+, which employs n-dimensional hypercubes for higher scalability.

4.1 MPV^0: The Strawman Construction

Before going deep into details, we introduce the following definitions closely related to the strawman construction.

Definition 1 (Ranked Object Set)**.** *Each key $k_s \in$ KEY is associated with a ranked object set $\mathrm{RO}_s = \{(k_s, v_c, t_c, c)\}_{c \in [d_s]}$, where (k_s, v_c, t_c, c), denoted by $\mathrm{RO}_s.c$, means that the object with value v_c and timestamp t_c is the c-th version for key k_s.*

Definition 2 (Verification Matrix)**.** *Each key $k_s \in$ KEY is associated with a $\mu_s \times \mu_s$ verification matrix $\mathbf{M}_s$, where the element at the i-th row and j-th column for $i, j \in [\mu_s]$ is defined as:*

$$\mathbf{M}_s[i][j] = \begin{cases} \mathrm{RO}_s.X, & if\ (i-1) \cdot \mu_s + j \leq d_s \\ \perp, & Otherwise \end{cases}$$

where $X = (i-1) \cdot \mu_s + j$. The set of elements in the i-th row and j-th column are denoted by $\mathbf{M}_s[i][*]$ and $\mathbf{M}_s[*][j]$, respectively.

The details of the strawman construction are shown in Protocol 1, where the matrix sizes $\{\mu_s\}_{s=1}^{N}$ are assumed publicly known. Our main idea is letting the on-chain ADS summarize the verifiable matrix for each key by using RSA accumulator, so that result soundness and completeness can be verified by testing if query results belong to the matrix or not. In particularly, the error locating process is similar to the two-dimensional parity-based method: The query result located at $\mathbf{M}_s[i][j]$ is deemed to be fake, if the verification regarding the i-th row and j-th column of $\mathbf{M}_s$ fails.

ADS Generation. Once enough data $\{(k_s, \mathsf{OBJ}_s)\}_{s=1}^{N}$ has been collected, the data owner uploads $\mathcal{ADS} = \{\Delta_s\}_{s=1}^{N}$ to the blockchain, where $\Delta_s = \{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i,j \in [\mu_s]}$ is a digest set summarizing the verifiable matrix $\mathbf{M}_s$. Specifically, $\delta_{\mathcal{L}_i} = \mathbf{acc}(\mathbf{M}_s[i][*])$ and $\delta_{\mathcal{C}_j} = \mathbf{acc}(\mathbf{M}_s[*][j])$ are the digest of i-th row and j-th column of $\mathbf{M}_s$, respectively.

VO Construction. Given a query $\mathcal{Q} = (k_s, [t_l, t_u])$, the SP first runs the Query algorithm (Alg. 1) to obtain the search results $\mathcal{SR}$ and the boundary results $\mathcal{BR}$, and constructs a verifiable matrix $\mathbf{M}_s$ according to Def. 1. Then, the SP takes $R = \mathcal{SR} \cup \mathcal{BR}$, $d = \mu_s$, and $n = 2$ as the input of the SetConstruct algorithm (Alg. 2) to obtain $(I, J, \{L_i\}_{i \in I}, \{C_j\}_{j \in J})$, i.e., the set of lines I and the set of columns J of matrix $\mathbf{M}_s$ that are covered by the query results R, and the subsects L_i and C_j of R that locate at the i-th line and the j-th column of matrix $\mathbf{M}_s$, s.t. $i \in I$ and $j \in J$, respectively. Specifically, for each ranked object $\mathbf{e} \in R$, Alg. 2 computes which line (resp. column) of matrix $\mathbf{M}_s$ object $\mathbf{e}$ locates in according to Def. 2, then inserts the line number i (resp. the column number j) into set I (resp. set J), and puts object $\mathbf{e}$ into set L_i and set C_j separately. Given the output of Alg. 2, the SP computes $\{(\pi_{\mathcal{L}_i}, \pi_{\mathcal{C}_j})\}_{i \in I, j \in J}$ in order to form the VO, where $\pi_{\mathcal{L}_i} = \mathbf{acc}(\mathbf{M}_s[i][*] - L_i)$ and $\pi_{\mathcal{C}_j} = \mathbf{acc}(\mathbf{M}_s[*][j] - C_j)$ are the proofs for the query results belonging to the i-th line and j-th column of matrix $\mathbf{M}_s$, respectively.

Verification. For the query $\mathcal{Q} = (k_s, [t_l, t_u])$, the user first checks if the query results $R = \mathcal{SR} \cup \mathcal{BR}$ meet the requirement of completeness: (1) The objects in R have continuous version numbers; (2) The objects in $\mathcal{SR}$ satisfy the query condition; (3) The objects in $\mathcal{BR}$ have the smallest and largest version numbers compared with those in $\mathcal{SR}$; (4) The objects in $\mathcal{BR}$ are outside the query range. Once the validation passes, the user runs $\mathsf{VerProof}(L_i, \pi_{\mathcal{L}_i}, \delta_{\mathcal{L}_i})$ for $i \in I$ to verify the authenticity of elements in L_i. Due to the security of RSA accumulator, algorithm VerProof outputs

1. $\mathcal{P}(x_i)$ denotes the prime number corresponding to element x_i. It can be implemented by a two-universal hash function [35].

Protocol 1 The Strawman Construction

ADS Generation (by the data owner)
Input: Key/objects (k_s, OBJ_s)
Output: Digest set Δ_s
Construct a ranked object set RO_s according to Def. 1
Construct a $\mu_s \times \mu_s$ matrix $\mathbf{M}_s$ according to Def. 2
for $i, j = 1$ **to** μ_s **do**
  $\delta_{\mathcal{L}_i} \leftarrow \mathsf{GenAcc}(\mathbf{M}_s[i][*])$; $\delta_{\mathcal{C}_j} \leftarrow \mathsf{GenAcc}(\mathbf{M}_s[*][j])$
$\Delta_s \leftarrow \{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i,j \in [\mu_s]}$

VO Construction (by the SP)
Input: Query $\mathcal{Q} = (k_s, [t_l, t_u])$, key/objects (k_s, OBJ_s)
Output: Search results $\mathcal{SR}$, verifiable object $\mathcal{VO}$
$(\mathcal{SR}, \mathcal{BR}) \leftarrow \mathsf{Query}(k_s, \mathsf{OBJ}_s, \mathcal{Q})$; $\{\Pi_{\mathcal{L}}, \Pi_{\mathcal{C}}\} \leftarrow \emptyset$
Construct a $\mu_s \times \mu_s$ matrix $\mathbf{M}_s$ according to Def. 2
$(I, J, \{L_i\}_{i \in I}, \{C_j\}_{j \in J}) \leftarrow \mathsf{SetConstruct}(\mathcal{SR} \cup \mathcal{BR}, \mu_s, 2)$
for each $i \in I$ **do**
  $\pi_{\mathcal{L}_i} \leftarrow \mathsf{GenProof}(L_i, \mathbf{M}_s[i][*])$; $\Pi_{\mathcal{L}} \leftarrow \Pi_{\mathcal{L}} \cup \pi_{\mathcal{L}_i}$
for each $j \in J$ **do**
  $\pi_{\mathcal{C}_j} \leftarrow \mathsf{GenProof}(C_j, \mathbf{M}_s[*][j])$; $\Pi_{\mathcal{C}} \leftarrow \Pi_{\mathcal{C}} \cup \pi_{\mathcal{C}_j}$
$\mathcal{VO} \leftarrow (\mathcal{BR}, \Pi_{\mathcal{L}}, \Pi_{\mathcal{C}})$

Verification (by the user)
Input: Search result $\mathcal{SR}$, verifiable object $\mathcal{VO}$, digest set Δ_s
Parse $\mathcal{VO}$ as $(\mathcal{BR}, \Pi_{\mathcal{L}}, \Pi_{\mathcal{C}})$; $(\mathrm{EL}, \mathrm{EC}) \leftarrow \emptyset$
$flag \leftarrow 0$ ▷ *0 indicates query verification fails*
$(I, J, \{L_i\}_{i \in I}, \{C_j\}_{j \in J}) \leftarrow \mathsf{SetConstruct}(\mathcal{SR} \cup \mathcal{BR}, \mu_s, 2)$
for each $i \in I$ **do**
  if $\mathsf{VerProof}(L_i, \pi_{\mathcal{L}_i}, \delta_{\mathcal{L}_i}) = 0$ **then**
    $\mathrm{EL} \leftarrow \mathrm{EL} \cup i$
if $\mathrm{EL} = \emptyset$ **then**
  $flag \leftarrow 1$
else
  for each $j \in J$ **do**
    if $\mathsf{VerProof}(C_j, \pi_{\mathcal{C}_j}, \delta_{\mathcal{C}_j}) = 0$ **then**
      $\mathrm{EC} \leftarrow \mathrm{EC} \cup j$
$\mathcal{VR} \leftarrow (flag, \mathrm{EL}, \mathrm{EC})$

Algorithm 1 $\mathsf{Query}(k_s, \mathsf{OBJ}_s, \mathcal{Q})$

$\mathcal{SR} \leftarrow \{\mathrm{RO}_s.X, \ldots, \mathrm{RO}_s.Y\}$ s.t. $t_X \geq t_l$, $t_{X-1} < t_l$, $t_Y \leq t_u$ and $t_{Y+1} > t_u$ ▷ $\mathcal{Q} = (k_s, [t_l, t_u])$
$\mathcal{BR} \leftarrow \{\mathrm{RO}_s.(X-1), \mathrm{RO}_s.(Y+1)\}$

Algorithm 2 SetConstruct(R, d, n)

for each element $\mathbf{e} = (k, v_X, t_X, X) \in R$ **do**
  if $n = 2$ **then**
    $i \leftarrow (X-1)/d + 1$; $j \leftarrow X - (i-1) \cdot d$;
    $L_i \leftarrow L_i \cup \mathbf{e}$; $I \leftarrow I \cup i$; $C_j \leftarrow C_j \cup \mathbf{e}$; $J \leftarrow J \cup j$
  else
    for $i = n$ to 2 **do**
      $j \leftarrow (X-1)/d^{(i-1)} + 1$; $X \leftarrow X - (j-1) \cdot d^{(i-1)}$;
      $F \leftarrow F \cup (i,j)$; $Z_{i,j} \leftarrow Z_{i,j} \cup \mathbf{e}$
    $j \leftarrow X$; $F \leftarrow F \cup (1,j)$; $Z_{i,j} \leftarrow Z_{1,j} \cup \mathbf{e}$
if $n = 2$ **then**
  return $(I, J, \{L_i\}_{i\in I}, \{C_j\}_{j\in J})$
else
  return $(F, \{Z_{i,j}\}_{(i,j)\in F})$

1 only when $L_i \subseteq \mathbf{M}_s[i][*]$. If all the lines in I pass the validation, this means that all the query results R are real, validating result soundness. Otherwise, the user locates fake results by using the set of lines EL and the set of columns EC failed validation, i.e., $0 \leftarrow \mathsf{VerProof}(L_i, \pi_{\mathcal{L}_i}, \delta_{\mathcal{L}_i})$ and $0 \leftarrow \mathsf{VerProof}(C_j, \pi_{\mathcal{C}_j}, \delta_{\mathcal{C}_j})$ for $i \in$ EL and $j \in$ EC.

Example. To illustrate, let us consider the following example: Assume that the data flow of key k_s is in the form of $\mathsf{OBJ}_s = \{(v_i, t_i)\}_{i=1}^{24}$ where $t_i = i + 100$. The verifiable matrix $\mathbf{M}_s$ with parameter $\mu_s = 5$ is as shown in Fig. 3, where the key k_s is omitted. Given a query $\mathcal{Q} = (k_s, [113, 114])$, the search results $\mathcal{SR} = \{(k_s, v_{13}, 113, 13), (k_s, v_{14}, 114, 14)\}$ and the boundary results $\mathcal{BR} = \{(k_s, v_{12}, 112, 12), (k_s, v_{15}, 115, 15)\}$ locate at line $I = \{3\}$ and columns $J = \{2, 3, 4, 5\}$ of matrix $\mathbf{M}_s$. The subset of $R = \mathcal{SR} \cup \mathcal{BR}$ locate at the 3rd line, and span from the 2nd column to the 5th column of matrix $\mathbf{M}_s$ (covered by blue rectangle): $L_3 = R$, $C_2 = \{(k_s, v_{12}, 112, 12)\}$, $C_3 = \{(k_s, v_{13}, 113, 13)\}$, $C_4 = \{(k_s, v_{14}, 114, 14)\}$, and $C_5 = \{(k_s, v_{15}, 115, 15)\}$. If the SP returns fake results $\mathcal{SR}' = \{(k_s, v'_{13}, 113, 13), (k_s, v_{14}, 114, 14)\}$, algorithms $\mathsf{VerProof}(L_3, \pi_{\mathcal{L}_3}, \delta_{\mathcal{L}_3})$ and $\mathsf{VerProof}(C_3, \pi_{\mathcal{C}_3}, \delta_{\mathcal{C}_3})$ output 0, and the user knows that the object located at $\mathbf{M}_s[3][3]$ is fake (the intersection of the yellow line and yellow column). If the SP returns incomplete results $\mathcal{SR}' = \{(k_s, v_{14}, 114, 14)\}$, it has to forge boundary results $\mathcal{BR}' = \{(k_s, v_{13}, 112', 13), (k_s, v_{15}, 115, 15)\}$ to convince the user that the objects in $\mathcal{BR}'$ are outside the query range. However, the user will know that the object located at $\mathbf{M}_s[3][3]$ is fake since the verification of $\mathbf{M}_s[3][*]$ and $\mathbf{M}_s[*][3]$ fails.

4.2 MPV$^+$: The Improved Construction

In the strawman construction, the digest set Δ_s associated with key k_s is of size $O(2\mu_s)$, where μ_s as the number of lines/columns in matrix $\mathbf{M}_s$ is determined by the data size d_s. For a large-scale dataset, the strawman construction incurs massive on-chain data, lacking scalability. To alleviate this problem, we extends the two-dimensional verifiable matrix to an n-dimensional verifiable hypercube, reducing the ADS size from $O(2\mu_s)$ to $O(n \cdot \nu_s)$, s.t. $\mu_s \approx \nu_s^{n/2}$[2]. Besides the ranked object set defined in Def. 1, the improved construction works under the following definitions:

Definition 3 (n-Integer Coordinate System). *$\mathbb{Z}^n = \{(x_1, \ldots, x_n)\}$ is an n-dimensional cartesian coordinate system in integer field, in which any point is represented by an n-dimensional coordinate $(x_1, \ldots, x_n)$ where $x_i \in \mathbb{Z}$ for $i \in [n]$.*

2. Given $d_s = |\mathsf{OBJ}_s|$, we have $\mu_s = \lceil d_s^{1/2} \rceil$ and $\nu_s = \lceil d_s^{1/n} \rceil$.

ln \ cn	1	2	3	4	5
1	(v_1,101,1)	(v_2,102,2)	(v_3,103,3)	(v_4,104,4)	(v_5,105,5)
2	(v_6,106,6)	(v_7,107,7)	(v_8,108,8)	(v_9,109,9)	(v_{10},110,10)
3	(v_{11},111,11)	(v_{12},112,12)	(v_{13},113,13)	(v_{14},114,14)	(v_{15},115,15)
4	(v_{16},116,16)	(v_{17},117,17)	(v_{18},118,18)	(v_{19},119,19)	(v_{20},120,20)
5	(v_{21},121,21)	(v_{22},122,22)	(v_{23},123,23)	(v_{24},124,24)	

Fig. 3: Illustrative example of verifiable matrix $\mathbf{M}_s$.

Definition 4 ($(n-1)$-Face). *An n-integer coordinate system $\mathbb{Z}^n$ is composed of a set of $(n-1)$-faces, denoted by $\{\mathcal{F}_{i,j}\}_{i\in[n],j\in\mathbb{Z}}$, where $\mathcal{F}_{i,j}$ as the j-th face at the i-th dimension contains a set of points represented by coordinates $\{(x_1, x_2, \ldots, x_n)\}_{x_i=j}$.*

Definition 5 (Verifiable Hypercube). *Each key $k_s \in \mathit{KEY}$ is associated with a $\underbrace{\nu_s \times \cdots \times \nu_s}_{n}$ verification hypercube $\mathbf{Q}_s$, where each element corresponds to an n-dimensional integer coordinate $(x_1, \ldots, x_n)$ for $x_i \in [\nu_s]$ and $i \in [n]$. The element at coordinate $(x_1, \ldots, x_n)$, denoted by $\mathbf{Q}_s[x_1, \ldots, x_n]$ is defined as:*

$$\mathbf{Q}_s[x_1, \ldots, x_n] = \begin{cases} \mathrm{RO}_s.X, \ if\ x_1 + \sum_{i=2}^{n}(x_i - 1) \cdot \nu_s^{i-1} \le d_s \\ \perp, \qquad\qquad Otherwise \end{cases}$$

where $X = x_1 + \sum_{i=2}^{n}(x_i - 1) \cdot \nu_s^{i-1}$. The set of elements located at face $\mathcal{F}_{i,j}$ is denoted by $\mathbf{Q}_s[i, j]$, where $i \in [n]$ and $j \in [\nu_s]$.

The details of the improved construction are shown in Protocol 2, where the hypercube sizes $\{\nu_s\}_{s=1}^N$ are assumed publicly known. The main difference from the strawman construction is that the ADS summarizes the verifiable hypercube for each key, so that result soundness and completeness can be verified by testing if query results belong to the hypercube or not. As for range query $\mathcal{Q} = (k_s, [t_l, t_u])$, the result located at $\mathbf{Q}_s[x_1, \ldots, x_n]$ is error, if the verification regarding faces $\mathcal{F}_{1,x_1}, \ldots, \mathcal{F}_{n,x_n}$ fails.

ADS Generation. Once the verifiable hypercube $\mathbf{Q}_s$ is built based on Def. 5, the data owner calculates $\Delta_s = \{\delta_{\mathcal{F}_{i,j}}\}_{i\in[n],j\in[\nu_s]}$ as the digest set of key k_s, where $\delta_{\mathcal{F}_{i,j}} = \mathbf{acc}(\mathbf{Q}_s[i, j])$ summarizes the elements located in face $\mathcal{F}_{i,j}$ of $\mathbf{Q}_s$. The on-chain ADS is set as $\mathcal{ADS} = \{\Delta_s\}_{s=1}^N$.

VO Construction. The SP first runs Alg. 1 to obtain the query results $R = \mathcal{SR} \cup \mathcal{BR}$ and constructs a verifiable hypercube $\mathbf{Q}_s$ according to Def. 5. Then, the SP takes R, $d = \nu_s$, and $n > 2$ as the input of Alg. 2 to get $(F, \{Z_{i,j}\}_{(i,j)\in F})$, i.e., the set of faces F of hypercube $\mathbf{Q}_s$ covered by the query results R, and the subsect $Z_{i,j}$ of R locating at face $\mathcal{F}_{i,j}$ of $\mathbf{Q}_s$, s.t. $(i, j) \in F$. Specifically, for each ranked object $\mathbf{e} \in R$, Alg. 2 calculates which $(n-1)$-faces of hypercube $\mathbf{Q}_s$ object $\mathbf{e}$ locates in according to Def. 5, then inserts the face number (i, j) into set F, and put object $\mathbf{e}$ into set $Z_{i,j}$. Given the output of Alg. 2, the SP computes $\{\pi_{\mathcal{F}_{i,j}}\}_{(i,j)\in F}$ to form the VO, where $\pi_{\mathcal{F}_{i,j}} = \mathbf{acc}(\mathbf{Q}_s[i, j] - Z_{i,j})$ is the proof for the results belonging to face $\mathcal{F}_{i,j}$ of hypercube $\mathbf{Q}_s$.

Verification. If the query results R meet the completeness requirement described in the strawman construction, the user runs $\mathsf{VerProof}(Z_{n,j}, \pi_{\mathcal{F}_{n,j}}, \delta_{\mathcal{F}_{n,j}})$ for $(n, j) \in F$ to authenticate the elements in $Z_{n,j}$. Due to the security of RSA accumulator, algorithm VerProof outputs 1 only when $Z_{n,j} \subseteq \mathbf{Q}_s[n, j]$. If the n-th dimensional faces in F pass the validation, this means that the query results R are authentic, validating result soundness. Otherwise, the fake results can

Protocol 2 The Improved Construction

ADS Generation (by the data owner)
Input: Key/objects (k_s, OBJ_s)
Output: Digest set Δ_s
1: Construct a ranked object set RO_s according to Def. 1
2: Construct a ν_s^n hypercube $\mathbf{Q}_s$ according to Def. 5
3: **for** $i = 1$ **to** n **do**
4: **for** $j = 1$ **to** ν_s **do**
5: $\delta_{\mathcal{F}_{i,j}} \leftarrow \mathsf{GenAcc}(\mathbf{Q}_s[i,j])$
6: $\Delta_s \leftarrow \{\delta_{\mathcal{F}_{i,j}}\}_{i\in[n],j\in[\nu_s]}$

VO Construction (by the SP)
Input: Query $\mathcal{Q} = (k_s, [t_l, t_u])$, key/objects (k_s, OBJ_s)
Output: Search results $\mathcal{SR}$, verifiable object $\mathcal{VO}$
1: $(\mathcal{SR}, \mathcal{BR}) \leftarrow \mathsf{Query}(k_s, \mathsf{OBJ}_s, \mathcal{Q})$; $\{\Pi_\mathcal{L}, \Pi_\mathcal{C}\} \leftarrow \emptyset$
2: Construct a ν_s^n hypercube $\mathbf{Q}_s$ according to Def. 5
3: $(F, \{Z_{i,j}\}_{(i,j)\in F}) \leftarrow \mathsf{SetConstruct}(\mathcal{SR} \cup \mathcal{BR}, \nu_s, n)$
4: **for** each $(n, j) \in F$ **do**
5: $\pi_{\mathcal{F}_{n,j}} \leftarrow \mathsf{GenProof}(Z_{n,j}, \mathbf{Q}_s[n,j])$; $\Pi_\mathcal{L} \leftarrow \Pi_\mathcal{L} \cup \pi_{\mathcal{F}_{n,j}}$
6: **for** each $(i, j) \in F \wedge i < n$ **do**
7: $\pi_{\mathcal{F}_{i,j}} \leftarrow \mathsf{GenProof}(Z_{i,j}, \mathbf{Q}_s[i,j])$; $\Pi_\mathcal{C} \leftarrow \Pi_\mathcal{C} \cup \pi_{\mathcal{F}_{i,j}}$
8: $\mathcal{VO} \leftarrow (\mathcal{BR}, \Pi_\mathcal{L}, \Pi_\mathcal{C})$

Verification (by the user)
Input: Search result $\mathcal{SR}$, verifiable object $\mathcal{VO}$, digest set Δ_s
1: Parse $\mathcal{VO}$ as $(\mathcal{BR}, \Pi_\mathcal{L}, \Pi_\mathcal{C})$; $(\mathrm{EL}, \mathrm{EC}) \leftarrow \emptyset$
2: $flag \leftarrow 0$ ▷ *0 indicates query verification fails*
3: $(F, \{Z_{i,j}\}_{(i,j)\in F}) \leftarrow \mathsf{SetConstruct}(\mathcal{SR} \cup \mathcal{BR}, \nu_s, n)$
4: **for** each $(n, j) \in F$ **do**
5: **if** $\mathsf{VerProof}(Z_{n,j}, \pi_{\mathcal{F}_{n,j}}, \delta_{\mathcal{F}_{n,j}}) = 0$ **then**
6: $\mathrm{EL} \leftarrow \mathrm{EL} \cup (n, j)$
7: **if** $\mathrm{EL} = \emptyset$ **then**
8: $flag \leftarrow 1$
9: **else**
10: **for** each $(i, j) \in F \wedge i < n$ **do**
11: **if** $\mathsf{VerProof}(Z_{i,j}, \pi_{\mathcal{F}_{i,j}}, \delta_{\mathcal{F}_{i,j}}) = 0$ **then**
12: $\mathrm{EC} \leftarrow \mathrm{EC} \cup (i, j)$
13: $\mathcal{VR} \leftarrow (flag, \mathrm{EL}, \mathrm{EC})$

be located by using the set of faces failed validation, i.e., $0 \leftarrow \mathsf{VerProof}(Z_{i,j}, \pi_{\mathcal{F}_{i,j}}, \delta_{\mathcal{F}_{i,j}})$ for $(i, j) \in \mathrm{EL} \cup \mathrm{EC}$.

Example. Let us return to the sample data stream of key k_s, i.e., $\mathsf{OBJ}_s = \{(v_i, i + 100)\}_{i=1}^{24}$. Taking $n = 3$ and $\nu_s = 3$ as input, the verifiable hypercube $\mathbf{Q}_s$ is constructed as shown in Fig. 4. Given a query $\mathcal{Q} = (k_s, [113, 114])$, the search results $\mathcal{SR} = \{(k_s, v_{13}, 113, 13), (k_s, v_{14}, 114, 14)\}$ and the boundary results $\mathcal{BR} = \{(k_s, v_{12}, 112, 12), (k_s, v_{15}, 115, 15)\}$ locate at faces $\mathcal{F}_{1,1}$, $\mathcal{F}_{1,2}$, $\mathcal{F}_{1,3}$, $\mathcal{F}_{2,1}$, $\mathcal{F}_{2,2}$, $\mathcal{F}_{3,2}$ of hypercube $\mathbf{Q}_s$ (covered by blue rectangle). The subsets of $\mathcal{SR} \cup \mathcal{BR}$ are $\{Z_{1,1}, Z_{1,2}, Z_{1,3}, Z_{2,1}, Z_{2,2}, Z_{3,2}\}$. If the SP returns fake results $\mathcal{SR}' = \{(k_s, v'_{13}, 113, 13), (k_s, v_{14}, 114, 14)\}$, algorithms $\mathsf{VerProof}(Z_{1,1}, \pi_{\mathcal{F}_{1,1}}, \delta_{\mathcal{F}_{1,1}})$, $\mathsf{VerProof}(Z_{2,2}, \pi_{\mathcal{F}_{2,2}}, \delta_{\mathcal{F}_{2,2}})$, and $\mathsf{VerProof}(Z_{3,2}, \pi_{\mathcal{F}_{3,2}}, \delta_{\mathcal{F}_{3,2}})$ will output 0, and the user will know that the results located at $\mathbf{Q}_s[1, 2, 2]$ is fake. If the SP returns incomplete results $\mathcal{SR}' = \{(k_s, v_{14}, 114, 14)\}$ and forges fake boundary results $\mathcal{BR}' = \{(k_s, v_{13}, 112', 13), (k_s, v_{15}, 115, 15)\}$, the user can still know that the object located at $\mathbf{Q}_s[1, 2, 2]$ is fake since the verification regarding to faces $\mathcal{F}_{1,1}$, $\mathcal{F}_{2,2}$, and $\mathcal{F}_{3,2}$ fails. Compared with the strawman construction, the size of digest set Δ_s is reduced from 10 to 9. When the data size d_s increases to 1 million, the improved version reduces the size of digest set from $2,000$ to 300, achieving a savings of 85% storage spaces compared with the strawman construction.

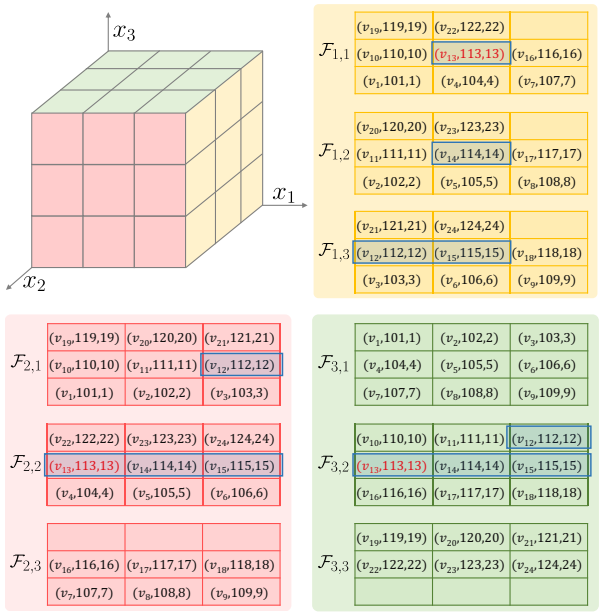


Fig. 4: Illustrative example of verifiable hypercube $\mathbf{Q}_s$.

5 The Gas-Efficient MPV Scheme

5.1 Main Idea

Our main idea is to trade higher off-chain computational costs for lower storage costs on the blockchain. The gas-efficient version is extended from the basic MPV scheme. For ease of illustration, we first provide a basic construction, denoted by MPV^0-GE, based on the verifiable matrices defined in Def. 2, and then we present an improved construction, denoted by MPV$^+$-GE, based on the n-dimensional verifiable hypercube defined in Def. 5. At a high-level view, for the verifiable matrix/hypercube of key k_s, the data owner uploads the digest of the digest set Δ_s, i.e., $\delta_s = \mathbf{acc}(\Delta_s)$, to the blockchain. In this way, the digest size of key k_s is reduced from $O(2\mu_s)$ or $O(n{\cdot}\nu_s)$ to $O(1)$. Instead of retrieving the line/column digests $\{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i,j\in[\mu_s]}$ or the face digests $\{\delta_{\mathcal{F}_{i,j}}\}_{i\in[n],j\in[\nu_s]}$ from the blockchain, the user obtains these information from the VO returned by the SP. By combining the on-chain ADS and the off-chain VO, the user performs the following two-stage verification:

- **Digest Verification.** The user authenticates whether all the digests returned by the SP are real or not.
- **Query Verification.** If the digest verification passes, the user authenticates query results as before; Otherwise, the query process is considered incredible.

5.2 The Detailed Construction of MPV^0-GE

As shown in Protocol 3 (excluding the boxed codes), MPV^0-GE is built based on the strawman construction MPV^0.

ADS Generation. The on-chain ADS is constructed as $\mathcal{ADS} = \{\delta_s\}_{s=1}^N$, where $\delta_s = \mathbf{acc}(\Delta_s)$ is the digest of the digest set Δ_s and $\Delta_s = \{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i,j\in[\mu_s]}$ contains the line/column digests of the verifiable matrix $\mathbf{M}_s$.

VO Construction. Compared with MPV^0, the VO additionally includes $\{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i\in I,j\in J}$, the digests for the lines/columns of matrix $\mathbf{M}_s$ covered by the query results and relative proofs $\{(\pi_i, \pi_j)\}_{i\in I,j\in J}$, where $\pi_i = \mathbf{acc}(\Delta_s - \delta_{\mathcal{L}_i})$ and $\pi_j = \mathbf{acc}(\Delta_s - \delta_{\mathcal{C}_j})$. Note that, once the digest sets $\{\Delta_s\}_{s=1}^N$ are computed, the SP can locally keep them to avoid repetitive calculations later.

Verification. The verification process consists of two stages: digest verification and query verification. At the first stage, the user runs algorithm $\mathsf{VerProof}(\delta_{\mathcal{L}_i}, \pi_i, \delta_s)$ and algorithm $\mathsf{VerProof}(\delta_{\mathcal{C}_j}, \pi_j, \delta_s)$ for $i \in I$ and $j \in J$ to verify if the digests returned by the SP are authentic. Due to the security of RSA accumulator, algorithm VerProof outputs 1 only when $\delta_{\mathcal{L}_i} \subseteq \Delta_s$ and $\delta_{\mathcal{C}_j} \subseteq \Delta_s$. If the first-stage validation passes, the user further authenticates the query results in the same way as the strawman construction.

5.3 The Detailed Construction of MPV$^+$-GE

Based on MPV$^+$, the detailed algorithms of MPV$^+$-GE are shown in Protocol 3 (including the boxed codes).

ADS Generation. The on-chain ADS is constructed as $\mathcal{ADS} = \{\delta_s\}_{s=1}^N$, where $\delta_s = \mathbf{acc}(\Delta_s)$ is the digest of the digest set Δ_s and $\Delta_s = \{\delta_{\mathcal{F}_{i,j}}\}_{i\in[n],j\in[\nu_s]}$ contains the face digests of the verifiable hypercube $\mathbf{Q}_s$.

VO Construction. Compared with MPV$^+$, the VO additionally includes $(\{\delta_{\mathcal{F}_{n,j}}\}_{(n,j)\in F}, \{\delta_{\mathcal{F}_{i,j}}\}_{(i,j)\in F\wedge i<n})$, the digests for the $(n-1)$-faces of hypercube $\mathbf{Q}_s$ covered by the query results and relative proofs $(\{\pi_{n,j}\}_{(n,j)\in F}, \{\pi_{i,j}\}_{(i,j)\in F\wedge i<n})$, where $\pi_{i,j} = \mathbf{acc}(\Delta_s - \delta_{i,j})$. Similarity, the SP can locally keep the digest sets $\{\Delta_s\}_{s=1}^N$ to save computational overheads.

Verification. The verification process consists of two stages: digest verification and query verification. At the first stage, the user runs algorithm $\mathsf{VerProof}(\delta_{\mathcal{F}_{i,j}}, \pi_{i,j}, \delta_s)$ for $(i,j) \in F$ to verify if the digests returned by the SP are authentic. Due to the security of RSA accumulator, algorithm VerProof outputs 1 only when $\delta_{\mathcal{F}_{i,j}} \subseteq \Delta_s$. If the first-stage validation passes, the user further authenticates the query results in the same way as MPV$^+$.

5.4 Constant On-Chain Storage Costs

In both MPV^0-GE and MPV$^+$-GE, each key is associated with a constant-size digest δ_s, and the on-chain ADS is of size $O(N)$ that is linear with the number of keys. By further refining the digests associated with the universal keys, the gas-efficient MPV scheme can be improved to achieve an constant-size on-chain ADS in the following way:

Let MPV* denote the constant-cost MPV scheme, and let $\mathrm{UD} = \{\delta_s\}_{s=1}^N$ denote the universal digest set. In ADS generation, the data owner uploads $\mathcal{ADS} = \delta_{\mathcal{U}}$ to the blockchain, where $\delta_{\mathcal{U}} = \mathbf{acc}(\mathrm{UD})$ is the digest of the universal digest set UD. Given a query $\mathcal{Q} = (k_s, [t_l, t_u])$ the SP needs to return the digest δ_s and a proof $\pi_s = \mathrm{acc}(\mathrm{UD} - \delta_s)$ besides the VO constructed in Protocol 3, so that the user first authenticates δ_s by running $\mathsf{VerProof}(\delta_s, \pi_s, \delta_{\mathcal{U}})$ before performing the two-stage verification process. Therefore, the constant-size on-chain ADS is achieved by the sacrifice of verification efficiency. In other word, the smaller the size of ADSs, the higher the verification costs.

To illustrate the differences of on-chain storage costs among our proposed constructions, we provide the sample ADS generation processes for MPV^0, MPV^0-GE, and MPV* in Fig. 5. Assume that the outsourced dataset is in the form of $\mathcal{D} = \{(k_s, \mathsf{OBJ}_s)\}_{s=1}^3$ with $|\mathsf{OBJ}_s| = 4$, and $\mu_s = 2$. In MPV^0, the data owner first constructs three ranked object sets, $\{\mathrm{RO}_s\}_{s=1}^3$, according to Def. 1, and then constructs three 2×2 verification matrixes, $\{\mathbf{M}_s\}_{s=1}^3$, according to Def. 2. For each line/column of verification matrix $\mathbf{M}_s$, the data owner runs algorithm GenAcc to generate a digest. That is to say, MPV^0 generates 4 line/column digests for each key k_s, i.e., $\Delta_s = \{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i,j\in[2]}$, and it will generates 12 digests in total, i.e., $\{\Delta_s\}_{s=1}^3$. In MPV^0-GE, the data owner first generates the digest sets, $\{\Delta_s\}_{s=1}^3$, in the same way as MPV^0. Then, for each digest set Δ_s, it performs algorithm GenAcc to generate a digest δ_s. Therefore, MPV^0-GE will generate 3 digests in total, i.e., $\mathrm{UD} = \{\delta_s\}_{s=1}^3$. In MPV*, the data owner first generates the digest set $\mathrm{UD} = \{\delta_s\}_{s=1}^3$ in the same way as MPV^0-GE. Then, for the digest set UD, it performs algorithm GenAcc to generate a digest $\delta_{\mathcal{U}}$. In this example, the number of digests in MPV^0, MPV^0-GE,

Protocol 3 The GAS-Efficient MPV Scheme

ADS Generation (by the data owner)
Input: Key/objects (k_s, OBJ_s)
Output: Digest of digest set δ_s
1: Execute lines 1-5 of ADS Generation in Protocol 1 to obtain a digest set $\Delta_s = \{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i,j\in[\mu_s]}$
Execute lines 1-6 of ADS Generation in Protocol 2 to obtain a digest set $\Delta_s \leftarrow \{\delta_{\mathcal{F}_{n,j}}\}_{i\in[n],j\in[\nu_s]}$
2: $\delta_s \leftarrow \mathsf{GenAcc}(\Delta_s)$

VO Construction (by the SP)
Input: Query $\mathcal{Q} = (k_s, [t_l, t_u])$, key/objects (k_s, OBJ_s)
Output: Search results $\mathcal{SR}$, verifiable object $\mathcal{VO}$
1: Execute lines 1-7 of VO Construction in Protocol 1 to obtain $(\mathcal{SR}, \mathcal{BR}, \Pi_{\mathcal{L}} = \{\pi_{\mathcal{L}_i}\}_{i\in I}, \Pi_{\mathcal{C}} = \{\pi_{\mathcal{C}_j}\}_{j\in J})$
2: Execute lines 1-5 of ADS Generation of Protocol 1 to obtain a digest set $\Delta_s = \{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i,j\in[\mu_s]}$; $(\Pi_{\mathcal{I}}, \Pi_{\mathcal{J}}) \leftarrow \emptyset$
3: **for** each $i \in I$ **do**
4: $\quad \pi_i \leftarrow \mathsf{GenProof}(\delta_{\mathcal{L}_i}, \Delta_s)$; $\Pi_{\mathcal{I}} \leftarrow \Pi_{\mathcal{I}} \cup (\delta_{\mathcal{L}_i}, \pi_i)$
5: **for** each $j \in J$ **do**
6: $\quad \pi_j \leftarrow \mathsf{GenProof}(\delta_{\mathcal{C}_j}, \Delta_s)$; $\Pi_{\mathcal{J}} \leftarrow \Pi_{\mathcal{J}} \cup (\delta_{\mathcal{C}_j}, \pi_j)$
Execute lines 1-7 of VO Construction in Protocol 2 to obtain $(\mathcal{SR}, \mathcal{BR}, \Pi_{\mathcal{L}} = \{\pi_{\mathcal{F}_{n,j}}\}_{(n,j)\in F}, \Pi_{\mathcal{C}} = \{\pi_{\mathcal{F}_{i,j}}\}_{(i,j)\in F\wedge i<n})$
Execute lines 1-6 of ADS Generation of Protocol 2 to obtain a digest set $\Delta_s \leftarrow \{\delta_{\mathcal{F}_{i,j}}\}_{i\in[n],j\in[\nu_s]}$
for each $(n,j) \in F$ **do**
$\quad \pi_{n,j} \leftarrow \mathsf{GenProof}(\delta_{\mathcal{F}_{n,j}}, \Delta_s)$; $\Pi_{\mathcal{I}} \leftarrow \Pi_{\mathcal{I}} \cup (\delta_{\mathcal{F}_{n,j}}, \pi_{n,j})$
for each $(i,j) \in F \wedge i < n$ **do**
$\quad \pi_{i,j} \leftarrow \mathsf{GenProof}(\delta_{\mathcal{F}_{i,j}}, \Delta_s)$; $\Pi_{\mathcal{J}} \leftarrow \Pi_J \cup (\delta_{\mathcal{F}_{i,j}}, \pi_{i,j})$
7: $\mathcal{VO} \leftarrow (\mathcal{BR}, \Pi_{\mathcal{L}}, \Pi_{\mathcal{C}}, \Pi_{\mathcal{I}}, \Pi_{\mathcal{J}})$

Verification (by the user)
Input: Search result $\mathcal{SR}$, verifiable object $\mathcal{VO}$, digest δ_s
Output: Verification report $\mathcal{VR}$
1: Parse $\mathcal{VO}$ as $(\mathcal{BR}, \Pi_{\mathcal{L}}, \Pi_{\mathcal{C}}, \Pi_{\mathcal{I}}, \Pi_{\mathcal{J}})$; $(\mathrm{EL}, \mathrm{EC}) \leftarrow \emptyset$
2: **for** each $(\delta_{\mathcal{L}_i}, \pi_i) \in \Pi_{\mathcal{I}}$ **do**
3: $\quad$ **if** $\mathsf{VerProof}(\delta_{\mathcal{L}_i}, \pi_i, \delta_s)$=0 **then**
4: $\quad\quad \mathrm{EL} \leftarrow \mathrm{EL} \cup i$
5: **for** each $(\delta_{\mathcal{C}_j}, \pi_j) \in \Pi_{\mathcal{J}}$ **do**
6: $\quad$ **if** $\mathsf{VerProof}(\delta_{\mathcal{C}_j}, \pi_j, \delta_s)$=0 **then**
7: $\quad\quad \mathrm{EC} \leftarrow \mathrm{EC} \cup j$
for each $(\delta_{\mathcal{F}_{n,j}}, \pi_{n,j}) \in \Pi_{\mathcal{I}}$ **do**
$\quad$ **if** $\mathsf{VerProof}(\delta_{\mathcal{F}_{n,j}}, \pi_{n,j}, \delta_s)$ =0
$\quad\quad \mathrm{EL} \leftarrow \mathrm{EL} \cup (n,j)$
for each $(\delta_{\mathcal{F}_{i,j}}, \pi_{i,j}) \in \Pi_{\mathcal{J}}$ **do**
$\quad$ **if** $\mathsf{VerProof}(\delta_{\mathcal{F}_{i,j}}, \pi_{i,j}, \delta_s)$ =0
$\quad\quad \mathrm{EC} \leftarrow \mathrm{EC} \cup (i,j)$
8: **if** $\mathrm{EL} = \emptyset \wedge \mathrm{EC} = \emptyset$ **then**
9: $\quad$ Execute lines 2-13 of Verification in Protocol 1
$\quad$ Execute lines 2-13 of Verification in Protocol 2
10: **else**
11: $\quad \mathcal{VR} \leftarrow (-1, \mathrm{EL}, \mathrm{EC})$ $\quad \triangleright$*-1 indicates digest verification fails*

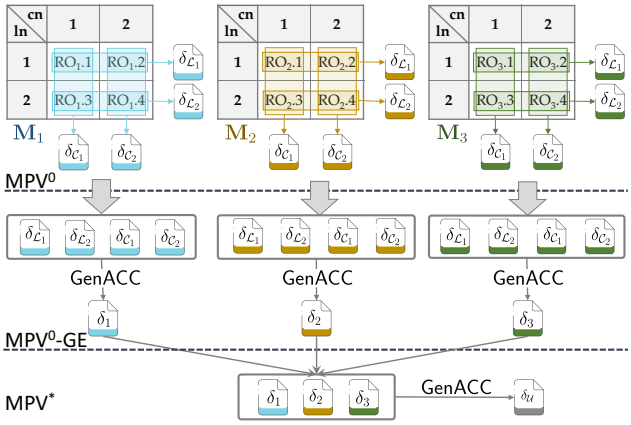


Fig. 5: Illustrative example of ADS generation processes.

and MPV* is 12, 3, and 1, respectively. As the scale of datasets increases, the differences of on-chain storage costs will be even greater. For example, for an outsourced dataset $\mathcal{D} = \{(k_s, \mathsf{OBJ}_s)\}_{s=1}^{10^3}$ with $|\mathsf{OBJ}_s| = 10^4$ and $\mu_s = 100$, the number of on-chain digests in MPV^0, MPV^0-GE, and MPV* will be 2×10^5, 10^3, and 1, respectively.

6 Security Analysis

6.1 Security Analysis of Basic MPV Scheme

As MPV^0 is a special case of MPV$^+$ (when the number of dimensions is set to 2), its security can be derived from that of MPV$^+$. Therefore, the following security analysis will focus on MPV$^+$.

Definition 6 (Security of MPV$^+$). *MPV$^+$ is considered secure if the probability for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ to succeed in the following experiment is negligible:*

- *Run the ADS generation algorithm in Protocol 2 to construct ADS, $\{\Delta_s\}_{s=1}^N$, and send corresponding dataset $\mathcal{D} = \{(k_s, \mathsf{OBJ}_s)\}_{s=1}^N$ to an adversary $\mathcal{A}$.*
- *$\mathcal{A}$ outputs a tuple of query $\mathcal{Q} = (k_s, [t_l, t_u])$, result $R = \mathcal{SR} \cup \mathcal{BR}$, and $\mathcal{VO} = (\mathcal{BR}, \Pi_\mathcal{L}, \Pi_\mathcal{C})$.*
- *$\mathcal{A}$ succeeds if $\mathcal{VO}$ passes the verification and satisfies the condition: $\{\mathbf{e}' \notin \mathrm{RO}_s \wedge \mathbf{e}' \in R\} \neq \varnothing \vee \{\mathbf{e} \in \mathcal{Q}(\mathrm{RO}_s) \wedge \mathbf{e} \notin R\} \neq \varnothing$, where $\mathcal{Q}(\mathrm{RO}_s)$ stands for the genuine search result on RO_s.*

The above definition guarantees that the probability, for a malicious SP in MPV$^+$ to convince the user with fake or incomplete results, is negligible. We now show that MPV$^+$ indeed satisfies the desired security requirement.

Theorem 1. *MPV$^+$ achieves result soundness and completeness, if hash functions are collision resistant, the RSA accumulator is secure, and the blockchain integrity is ensured.*

Proof. The security of MPV$^+$ is proven by contradiction.

Case 1: $\{\mathbf{e}' \notin \mathrm{RO}_s \wedge \mathbf{e}' \in R\} \neq \varnothing$. This means that adversary $\mathcal{A}$ forges query results R' by replacing an element $\mathbf{e}$ in query results $R = \mathcal{SR} \cup \mathcal{BR}$ with a fake element $\mathbf{e}'$ outside the ranked object set RO_s to compromise result soundness. Suppose that $\mathbf{e}'$ locates at face $\mathcal{F}_{n,x}$ of the hypercube $\mathbf{Q}_s$. Let $Z'_{n,x}$ denote the subset of fake query results R' locating at face $\mathcal{F}_{n,x}$ of the hypercube $\mathbf{Q}_s$. The adversary needs to forge a proof $\pi'_{\mathcal{F}_{n,x}}$ so that $\mathsf{VerProof}(Z'_{n,x}, \pi'_{\mathcal{F}_{n,x}}, \delta_{\mathcal{F}_{i,j}})$ outputs 1. Note that $\mathsf{H}(\mathbf{e}) \neq \mathsf{H}(\mathbf{e}')$ when $\mathbf{e} \neq \mathbf{e}'$ due to the collision resistance of hash functions, and that the on-chain ADS, $\{\delta_{\mathcal{F}_{i,j}}\}_{i\in[n],j\in[\nu_s]}$, is synchronized with the blockchain network. Therefore, a forged VO that makes the user accept the fake result, i.e., enabling algorithm VerProof to output 1, contradicts with the security of RSA accumulator.

Case 2: $\{\mathbf{e} \in \mathcal{Q}(\mathrm{RO}_s) \wedge \mathbf{e} \notin R\} \neq \varnothing$. This means that adversary $\mathcal{A}$ forges query results R' by skipping an element $\mathbf{e}$ in query results $R = \mathcal{SR} \cup \mathcal{BR}$ to compromise result completeness. Before the verification, the user will check if the query results $R = \mathcal{SR} \cup \mathcal{BR}$ meet the requirement of completeness: (1) The objects in R have continuous version numbers; (2) The objects in $\mathcal{SR}$ satisfy the query condition; (3) The objects in $\mathcal{BR}$ have the smallest and largest version numbers compared with those in $\mathcal{SR}$; (4) The objects in $\mathcal{BR}$ are outside the query range. To meet the above requirements, adversary $\mathcal{A}$ needs to add a fake element outside the ranked object set RO_s to make the version numbers in query results R' continuous. As proven in Case 1, it is impossible for the adversary to forge fake result passing soundness verification, validating result completeness. □

6.2 Security Analysis of GAS-Efficient MPV Scheme

Like basic scheme, we will focus on the security analysis of MPV$^+$-GE since MPV^0-GE is a special case of MPV$^+$-GE.

Definition 7 (Security of MPV$^+$-GE). *MPV$^+$-GE is considered secure if the probability for any PPT adversary $\mathcal{A}$ to succeed in the following experiment is negligible:*

- *Run the ADS generation algorithm in Protocol 3 to construct ADS, $\{\delta_s\}_{s=1}^N$, and send corresponding dataset $\mathcal{D} = \{(k_s, \mathsf{OBJ}_s)\}_{s=1}^N$ to an adversary $\mathcal{A}$.*
- *$\mathcal{A}$ outputs a tuple of query $\mathcal{Q} = (k_s, [t_l, t_u])$, result $R = \mathcal{SR} \cup \mathcal{BR}$, and $\mathcal{VO} = (\mathcal{BR}, \Pi_\mathcal{L}, \Pi_\mathcal{C}, \Pi_\mathcal{I}, \Pi_\mathcal{J})$.*
- *$\mathcal{A}$ succeeds if $\mathcal{VO}$ passes the verification and satisfies the condition: $\{(\delta'_{\mathcal{F}_{n,j}}, \pi'_{n,j}) \in \Pi_\mathcal{I} \wedge \delta'_{\mathcal{F}_{n,j}} \notin \Delta_s\} \neq \varnothing \vee \{(n,j) \in \mathcal{F} \wedge (\delta_{\mathcal{F}_{n,j}}, \pi_{n,j}) \notin \Pi_\mathcal{I}\} \neq \varnothing \vee \{\mathbf{e}' \notin \mathrm{RO}_s \wedge \mathbf{e}' \in R\} \neq \varnothing \vee \{\mathbf{e} \in \mathcal{Q}(\mathrm{RO}_s) \wedge \mathbf{e} \notin R\} \neq \varnothing$, where $\mathcal{Q}(\mathrm{RO}_s)$ stands for the genuine search result on RO_s.*

The above definition guarantees that the probability, for a malicious SP in MPV$^+$-GE to convince the user with fake or incomplete digests/results, is negligible. We now show that MPV$^+$-GE satisfies the desired security requirement.

Theorem 2. *MPV$^+$-GE achieves result soundness and completeness, if hash functions are collision resistant, the RSA accumulator is secure, and the blockchain integrity is ensured.*

Proof. MPV$^+$-GE built based on MPV$^+$ consists of a two-stage verification process, where the query verification process is performed in the same way as MPV$^+$. Thus, the impossibility of Case 1 ($\{\mathbf{e}' \notin \mathrm{RO}_s \wedge \mathbf{e}' \in R\} \neq \varnothing$) and Case 2 ($\{\mathbf{e} \in \mathcal{Q}(\mathrm{RO}_s) \wedge \mathbf{e} \notin R\} \neq \varnothing$) can be proven according to the same processes in Case 1 and Case 2 of Theorem 1, respectively. Next, we focus on the security of the digest verification process that can also be proven by contradiction.

Case 3: $\{(\delta'_{\mathcal{F}_{n,j}}, \pi'_{n,j}) \in \Pi_\mathcal{I} \wedge \delta'_{\mathcal{F}_{n,j}} \notin \Delta_s\} \neq \varnothing$. This means that adversary $\mathcal{A}$ forges set $\Pi'_\mathcal{I}$ by replacing a digest/proof pair $(\delta'_{\mathcal{F}_{n,j}}, \pi'_{n,j}) \in \Pi_\mathcal{I}$ with a forged pair $(\delta'_{\mathcal{F}_{n,j}}, \pi'_{n,j})$. Note that the digests $\{\delta_s\}_{s=1}^N$ of the digest sets $\{\Delta_s\}_{s=1}^n$ are stored as the on-chain ADS, which is synchronized with the blockchain network. Due to the

collision resistance of hash functions, $\mathsf{H}(\delta_{\mathcal{F}_{n,j}}) \neq \mathsf{H}(\delta'_{\mathcal{F}_{n,j}})$ when $\delta_{\mathcal{F}_{n,j}} \neq \delta'_{\mathcal{F}_{n,j}}$. The forged digest/proof pair enabling algorithm $\mathsf{VerProof}(\delta'_{\mathcal{F}_{n,j}}, \pi'_{n,j}, \delta_s)$ to output 1 implies a collision to the security of RSA accumulator.

Case 4: $\{(n,j) \in \mathcal{F} \wedge (\delta_{\mathcal{F}_{n,j}}, \pi_{n,j}) \notin \Pi_{\mathcal{I}}\} \neq \varnothing$. This means that adversary $\mathcal{A}$ forges a set $\Pi'_{\mathcal{I}}$ by removing a valid digest $\delta_{\mathcal{F}_{n,j}}$ from $\Pi_{\mathcal{I}}$. In this condition, the digest verification are passed since the returned digest/proof pairs are authentic. However, the query results originally located at face $\mathcal{F}_{n,j}$ need to be removed as $\delta_{\mathcal{F}_{n,j}}$ is omitted from $\Pi_{\mathcal{I}}$. That is, in the query verification process, adversary $\mathcal{A}$ needs to replace the elements located at $\mathcal{F}_{n,j}$ with some fake elements to make the version numbers in query results continuous. As proven in the Case 1 of Theorem 1, it is impossible for the adversary to forge fake result passing soundness verification, thereby proving that incomplete digests cannot pass the two-stage verification. □

7 Discussions

7.1 How to Support Dynamic Updates

Let $\{(k_s, \mathsf{OBJ}_s)\}_{s=1}^N$ and $\{(k_s, \mathsf{OBJ}'_s)\}_{s=1}^N$ denote the current outsourced dataset and the newly collected data, respectively where $|\mathsf{OBJ}_s| = d_s$ and $|\mathsf{OBJ}'_s| = d'_s$. To ensure the SP faithfully updates the outsrouced dataset, the data owner needs to update the on-chain ADS appropriately. In MPV^+, the ADS is in the form of $\mathcal{ADS} = \{\Delta_s\}_{s=1}^N$. To update the digest set Δ_s, the data owner sorts objects in OBJ'_s by the ascending order of their timestamps and constructs a new ranked object set $\mathrm{RO}'_s = \{\mathrm{RO}_s.(d_s+1), \ldots, \mathrm{RO}_s.(d_s+d'_s)\}$, where $\mathrm{RO}_s.(d_s+i)$ is the i-th version in OBJ'_s for $i \in [d'_s]$. Then, the data owner runs algorithm $\mathsf{SetConstruct}$ to obtain a set of faces F of hypercube $\mathbf{Q}_s$ covered by these new ranked objects RO'_s, and the subset $Z_{i,j}$ of RO'_s locating at face $\mathcal{F}_{i,j}$ of $\mathbf{Q}_s$, s.t. $(i,j) \in F$. After retrieving the original digests $\{\delta_{\mathcal{F}_{i,j}}\}_{(i,j)\in F}$ from the blockchain (if the digest is calculated for the first time, $\delta_{\mathcal{F}_{i,j}}$ is set to the group generator), the data owner generates the updated digests as $\delta'_{\mathcal{F}_{i,j}} \leftarrow \delta_{\mathcal{F}_{i,j}}^{\prod_{x\in Z_{i,j}} \mathcal{P}(\mathsf{H}(x))} \mod \mathbf{N}$ for $(i,j) \in F$.

MPV^0 is a special case of MPV^+, where the ADS can be updated in a similar way except that algorithm $\mathsf{SetConstruct}$ is run to obtain a set of lines I and a set of columns J of matrix $\mathbf{M}_s$ covered by the new ranked objects RO'_s and the subsets L_i and C_j of RO'_s that locate at the i-th line and the j-th column of matrix $\mathbf{M}_s$, respectively, s.t. $i \in I$ and $j \in J$. Given the on-chain digests $\{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i\in I, j\in J}$, the data owner updates the digests as $\delta'_{\mathcal{L}_i} \leftarrow \delta_{\mathcal{L}_i}^{\prod_{x\in L_i} \mathcal{P}(\mathsf{H}(x))} \mod \mathbf{N}$ and $\delta'_{\mathcal{C}_i} \leftarrow \delta_{\mathcal{C}_i}^{\prod_{x\in C_j} \mathcal{P}(\mathsf{H}(x))} \mod \mathbf{N}$ for $i \in I$ and $j \in J$.

The gas-efficient version is built based on the basic MPV scheme. The ADS is in the form of $\mathcal{ADS} = \{\delta_s\}_{s=1}^N$, where δ_s as the digest of the digest set Δ_s can be updated as follows: The data owner performs in the same way as MPV^0 (resp. MPV^+) to obtain $(I, J, \{L_i\}_{i\in I}, \{C_j\}_{j\in J})$ (resp. $(F, \{Z_{i,j}\}_{(i,j)\in F})$), and then retrieves the digests $\{(\delta_{\mathcal{L}_i}, \delta_{\mathcal{C}_j})\}_{i\in I, j\in J}$ (resp. $\{\delta_{\mathcal{F}_{i,j}}\}_{(i,j)\in F}$) from the SP while getting the digest δ_s from the blockchain. Once all the digests returned by the SP pass validation, the data owner calculates the updated digests $\{(\delta'_{\mathcal{L}_i}, \delta'_{\mathcal{C}_j})\}_{i\in I, j\in J}$ or $\{\delta'_{\mathcal{F}_{i,j}}\}_{(i,j)\in F}$ as the basic scheme, and update the digest as:

$$\delta'_s = \delta_s^{y/x \mod \varphi(\mathbf{N})} \mod \mathbf{N}. \quad (1)$$

For MPV^0-GE, $y = \prod_{i\in I} \mathcal{P}(\mathsf{H}(\delta'_{\mathcal{L}_i})) \cdot \prod_{j\in J} \mathcal{P}(\mathsf{H}(\delta'_{\mathcal{C}_j}))$ and $x = \prod_{i\in I} \mathcal{P}(\mathsf{H}(\delta_{\mathcal{L}_i})) \cdot \prod_{j\in J} \mathcal{P}(\mathsf{H}(\delta_{\mathcal{C}_j}))$; For MPV^+-GE, $y = \prod_{(i,j)\in F} \mathcal{P}(\mathsf{H}(\delta'_{\mathcal{F}_{i,j}}))$ and $x = \prod_{(i,j)\in F} \mathcal{P}(\mathsf{H}(\delta_{\mathcal{F}_{i,j}}))$.

7.2 How to Ensure Freshness Authentication

As for the outsourced key-value store subjected to continuous updates, each key is associated with multiple versions of objects, thus it is essential to ensure result freshness, besides result soundness and completeness. Specifically, we mainly consider the most commonly studied query type: freshness query $\mathcal{Q} = (k, [*, t_u])$ or $\mathcal{Q} = (k, [*])$ that retrieves the most recent object for key k as of t_u or so far. Since the freshness query produces only one matched result, the user only needs to verify result soundness and freshness without requiring error locating. Note that the freshness query $\mathcal{Q} = (k, [*, t_u])$ is actually a special case of the range query $\mathcal{Q} = (k, [t_l, t_u])$ (a range query without lower boundary). Hence, result freshness is assured or violated, depending on if query results pass completeness validation or not. As result soundness can be achieved by directly using our MPV schemes, we concentrate on authenticating result freshness for query $\mathcal{Q} = (k, [*])$ and provide the following definition:

Definition 8 (Fresh Version Set)**.** *Each key $k_s \in$ KEY is associated with a fresh version number d_s. The fresh version set is defined as* $\mathrm{FV} = \{(k_s, d_s)\}_{s=1}^N$*, where (k_s, d_s), denoted by* $\mathrm{FV}.s$*, means the latest version number of k_s is d_s.*

Our main idea is to incorporate the digest of the fresh version set, i.e., $\delta_F = \mathbf{acc}(\mathrm{FV})$, into the on-chain ADS so that the user can authenticate result freshness by testing if the version number in search results belongs to set FV or not. Specifically, in ADS generation, the data owner generates the digest by calculating $\delta_{\mathcal{F}} \leftarrow \mathsf{GenAcc}(\mathrm{FV})$. In VO construction, given a query $\mathcal{Q} = (k_s, [*])$, the SP puts the latest element $(k_s, v_{d_s}, t_{d_s}, d_s)$ in the ranked object set RO_s into the search result $\mathcal{SR}$, and generates a proof by calculating $\pi_{\mathcal{F}} \leftarrow \mathsf{GenProof}((k_s, d_s), \mathrm{FV})$. In verification, the user retrieves the digest $\delta_{\mathcal{F}}$ from the blockchain and authenticates result freshness by running $\mathsf{VerProof}((k_s, d'_s), \pi_{\mathcal{F}}, \delta_{\mathcal{F}})$, where d'_s is the version number in $\mathcal{SR}$. Due to the security of RSA accumulator, algorithm $\mathsf{VerProof}$ outputs 1 only when $(k_s, d'_s) \subseteq \mathrm{FV}$. This means that $d'_s = d_s$ is the fresh version number of key k_s, validating result freshness.

Complexity. Let α be the size of query results and let F_n be the set of n-th dimensional faces at which the query results locate in hypercube $\mathbf{Q}_s$. The computational cost for the data owner to generate a constant-size digest $\delta_{\mathcal{F}}$ is $O(N)$. As for a freshness query, there are only one search result and one or zero boundary result. Therefore, we have $|I| \leq 2$, $|F_n| \leq 2$, and $\alpha = 2$ when $\mathcal{Q} = (k_s, [*, t_u])$ and $|I| = 1$, $|F_n| = 1$, and $\alpha = 1$ when $\mathcal{Q} = (k_s, [*])$. For the first case, MPV^0 requires the SP to calculate a proof for each covered line, incurring computational costs $O(2\mu_s)$; MPV^+ needs to calculate a proof for the covered face at the n-th dimension incurring computational costs $O(2\nu_s^{n-1})$; MPV^0-GE and MPV^+-GE requires an extra proof for the digest associated with each covered line and each covered face incurring computational costs $O(4\mu_s)$ and $O(4\nu_s^{n-1})$, respectively. By contrast, the second case requires the SP to calculate an extra proof for the fresh version number, incurring computational costs $O(\mu_s + N)$, $O(\nu_s^{n-1} + N)$,

TABLE 3: Performance Comparison of MPV Schemes

	MPV^0	MPV$^+$	MPV^0-GE	MPV$^+$-GE
Data owner	$O(2\sum_{s=1}^{N} d_s)$	$O(n \cdot \sum_{s=1}^{N} d_s)$	$O(2\sum_{s=1}^{N}(d_s+\mu_s))$	$O(n \cdot \sum_{s=1}^{N}(d_s+\nu_s))$
SP	$O((\lvert I\rvert+\lvert J\rvert)\cdot\mu_s-2\alpha)$	$O(\lvert F\rvert\cdot\nu_s^{n-1}-n\cdot\alpha)$	$O(2(\lvert I\rvert+\lvert J\rvert)\cdot\mu_s-2\alpha)$	$O(2\lvert F\rvert\cdot\nu_s^{n-1}-n\cdot\alpha)$
User	$O((\lvert I\rvert+\lvert J\rvert)\cdot\mu_s)$	$O(\lvert F\rvert\cdot\nu_s^{n-1})$	$O((\lvert I\rvert+\lvert J\rvert)\cdot(\mu_s+1))$	$O(\lvert F\rvert\cdot(\nu_s^{n-1}+1))$
ADS Size	$O(2\sum_{s=1}^{N}\mu_s)$	$O(n\cdot\sum_{s=1}^{N}\nu_s)$	$O(N)$	$O(N)$
VO Size	$O(\lvert I\rvert+\lvert J\rvert)$	$O(\lvert F\rvert)$	$O(2\lvert I\rvert+2\lvert J\rvert)$	$O(2\lvert F\rvert)$

α is the size of query results; d_s is the data size of key k_s; N is the number of keys; n is the number of dimensions; α is the size of query results; μ_s and ν_s are the sizes of matrix $\mathbf{M}_s$ and hypercube $\mathbf{Q}_s$, respectively; I and J are the set of lines and columns of $\mathbf{M}_s$ covered by the query results, respectively; F is the set of faces of $\mathbf{Q}_s$ covered by the query results.

$O(2\mu_s+N)$, and $O(2\nu_s^{n-1}+N)$ in MPV^0, MPV$^+$, MPV^0-GE and MPV$^+$-GE, respectively. In both cases, the VO size as well as the verification cost incurred in all our schemes are constant.

Handling Updates. To update the on-chain digest $\delta_{\mathcal{F}}$, the data owner first requests the fresh version set $\mathrm{FV} = \{(k_s, d_s)\}_{s=1}^{N}$ from the SP and tests if $\mathbf{acc}(\mathrm{FV}) = \delta_{\mathcal{F}}$ or not. If so, this means that all the elements in set FV are correct. The data owner then calculates $d_s \leftarrow d_s + d'_s$, constructs a new fresh version set $\mathrm{FV}' = \{(k_s, d_s)\}_{s=1}^{N}$, and generates the updated digest as $\delta'_{\mathcal{F}} \leftarrow \mathsf{GenAcc}(\mathrm{FV}')$.

8 Evaluation

8.1 Performance Analysis

The performance is analyzed in terms of computational and communication costs incurred at different stages of our MPV schemes. As for computational costs, we only consider the expensive group operations related to RSA accumulator. Specifically, algorithm GenAcc requires performing power operations for $|X|$ times to generate a digest for a set X; Algorithm GenProof requires performing power operations for $|X-Y|$ times to generate a proof for the subset Y of set X; Algorithm VerProof requires performing power operations for $|Y|$ times to verify if $Y \subseteq X$. As for communication costs, we mainly consider the sizes of on-chain ADS and off-chain VO. The comparison results are shown in Table 3.

ADS Generation. In MPV^0 and MPV$^+$, the data owner generates a digest set for the verifiable matrix/hypercube associated with each key, incurring the total computational costs $O(2\sum_{s=1}^{N} d_s)$ and $O(n \cdot \sum_{s=1}^{N} d_s)$, respectively. Compared with the basic MPV scheme, the gas-efficient version requires calculating an extra digest for the digest set associated with each key. Thus, the total computational costs in MPV^0-GE and MPV$^+$-GE are $O(2\sum_{s=1}^{N}(d_s+\mu_s))$ and $O(n \cdot \sum_{s=1}^{N}(d_s+\nu_s))$, respectively. The ADS sizes in MPV^0, MPV$^+$, and the gas-efficient version are $O(2\sum_{s=1}^{N}\mu_s)$, $O(n \cdot \sum_{s=1}^{N}\nu_s)$, and $O(N)$, respectively, s.t. $\mu_s \approx \nu_s^{n/2}$.

VO Construction. For a range query that produces α matched results, MPV^0 requires the SP to calculate a proof for each covered line/column, incurring computational costs $O((|I|+|J|)\cdot\mu_s - 2\alpha)$ and VO size $O(|I|+|J|)$; MPV$^+$ needs to calculate a proof for each covered face incurring computational costs $O(|F|\cdot\nu_s^{n-1} - n\cdot\alpha)$ and VO size $O(|F|)$; Without considering the computational costs of digest sets, MPV^0-GE requires an extra proof for the digest associated with each covered line/column incurring computational costs $O(2(|I|+|J|)\cdot\mu_s - 2\alpha)$ and VO size $O(2|I|+2|J|)$; MPV$^+$-GE requires an extra proof for the digest associated with each covered face incurring computational costs $O(2|F|\cdot\nu_s^{n-1} - n\cdot\alpha)$ and VO size $O(2|F|)$.

Verification. For a range query with α matched results, the user needs to authenticate query results and locate errors, incurring the total computational costs $O((|I|+|J|)\cdot\mu_s)$ and $O(|F|\cdot\nu_s^{n-1})$ in MPV^0 and MPV$^+$, respectively. MPV^0-GE and MPV$^+$-GE require the user to perform the two-stage verification, resulting in computational costs are $O((|I|+|J|)\cdot(\mu_s+1))$ and $O(|F|\cdot(\nu_s^{n-1}+1))$, respectively.

8.2 Parameter Settings and Datasets

In our evaluation, the SP is simulated at a server with an Intel Xeon Gold 2.30GHz CPU and 128GB RAM. The data owner and user are set up on a portable laptop with an Intel Core i7 2.30GHz CPU and 8GB RAM. The experiment codes are implemented by Java, and the Ethereum test network[3] with smart contracts is used to simulate the process of uploading on-chain ADSs. For the cryptographic algorithms, we set the security parameter λ to 256, and apply SHA-256 to implement hash functions. Besides, the JPBC library is employed for group and bilinear pairing calculations. When implementing RSA accumulator with JPBC library, the costs of group-related operations mainly come from mapping accumulative values to integers, and thus the execution time incurred on each entity is determined by the running times of relevant algorithms. To verify the effectiveness of MPV, we select two real datasets for performance evaluation:

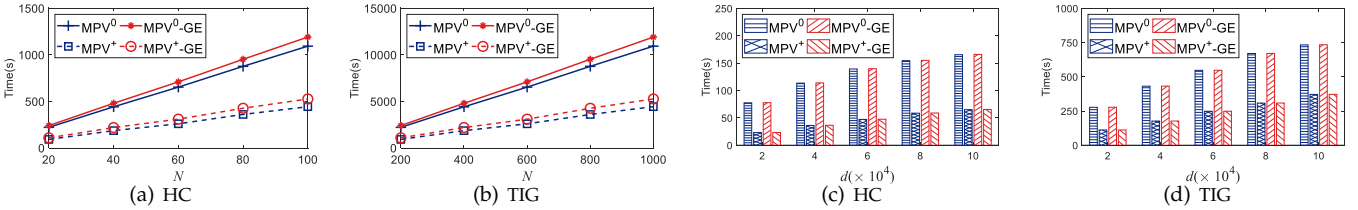
- Smart home dataset with weather information (HC for short)[4]. This dataset contains the readings of house appliances from a smart meter during 350 days and the weather conditions of the particular region. The data scale is about $100,000$ rows and each object is represented as (*key*, [*electricity consumption*, *temperature*], *timestamp*).
- Temperature IoT on Google cloud platform (TIG for short)[5]. This dataset contains $1,050,000$ rows of raw temperature data streaming from different sensors. Each object is represented as (*key*, [*temp_f*, *temp_c*, *device_id*], *timestamp*).

According to the analysis results in section 8.1, we can get that the number of keys N and the data size d of each key are two important parameters for the performance of our schemes. To test the parameter N, we preprocess the datasets so that each key contains 1,000 data and set N to $\{20, 40, 60, 80, 100\}$ in HC and $\{200, 400, 600, 800, 1,000\}$ in TIG, respectively. To show the impact of parameter d, we preprocess the datasets and set the range of the data size for a single key to $[2\times10^4,\ 10^5]$ in HC, and $[2\times10^5,\ 10^6]$ in TIG. Meanwhile, the number of dimensions in MPV$^+$ and MPV$^+$-GE is set to $n=3$; the query ratio $\tau = \alpha/d$ means the ratio of the result size α to the data size d and is set to $\{5\%, 10\%, 15\%, 20\%, 25\%\}$; the error rate of query results is fixed to 1%. The performance is evaluated by the following

3. https://rinkeby.etherscan.io/

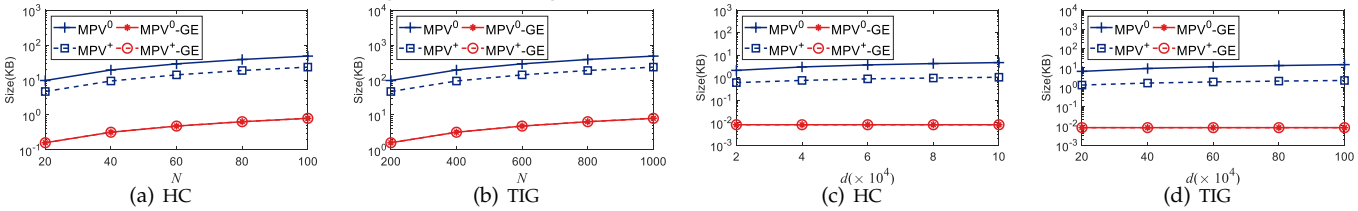
4. https://www.kaggle.com/datasets/taranvee/smart-home-dataset-with-weather-information

5. https://www.kaggle.com/datasets/mattpo/temperature-iot-on-gcp



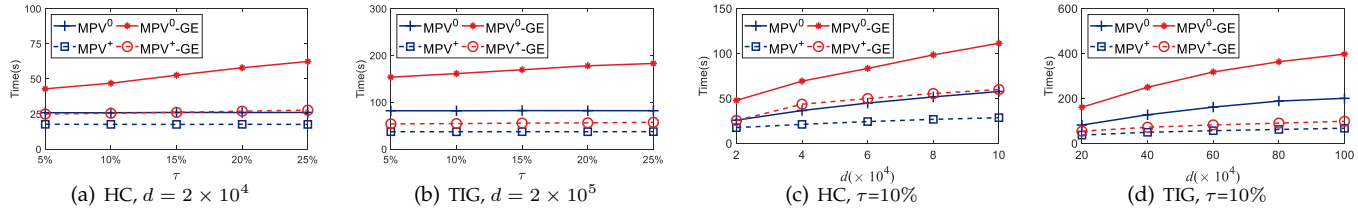
(a) HC (b) TIG (c) HC (d) TIG

Fig. 6: The ADS generation time on the data owner side.



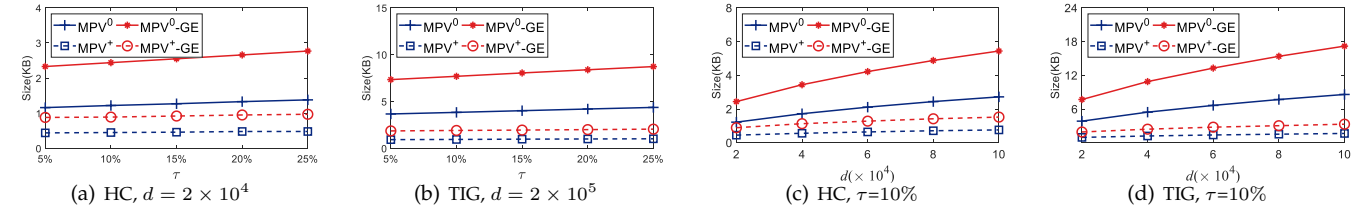
(a) HC (b) TIG (c) HC (d) TIG

Fig. 7: The storage costs on the blockchain.



(a) HC, $d = 2 \times 10^4$ (b) TIG, $d = 2 \times 10^5$ (c) HC, τ=10% (d) TIG, τ=10%

Fig. 8: The time of building VO on the SP side.



(a) HC, $d = 2 \times 10^4$ (b) TIG, $d = 2 \times 10^5$ (c) HC, τ=10% (d) TIG, τ=10%

Fig. 9: The size of VOs transmitted from the SP to the user.

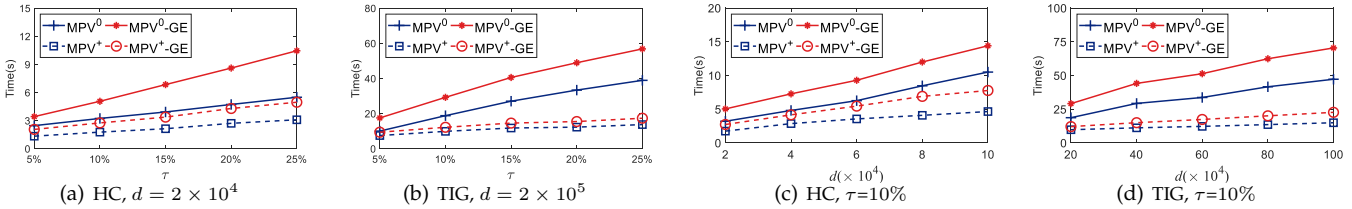
metrics: (1) ADS generation time; (2) The storage cost on blockchain; (3) VO construction time; (4) VO size; (5) Query authentication time; (6) Error locating time. The 1st metric is performed on the data owner side, and the 2nd metric is the on-chain storage overhead. The 3th and 4th metrics are executed on the SP side, and the 5th and 6th metrics are executed on the user side. To minimize deviation, each metric is run at least 100 times to obtain the average values.

8.3 Experimental results

ADS Generation. From Fig. 6, we can see that the ADS generation time of all schemes increases as the parameters N and d increase. The reason is obvious, the bigger values of N and d means that the more data needs to be processed to generate the ADSs, hence incurring more execution time. For the same reason, the ADS generation time of all schemes on TIG is longer than that on HC under the same parameter settings. In terms of four schemes, MPV$^+$ performs the best, and MPV^0-GE performs the worst. For example, when $d = 4 \times 10^4$, the ADS generation time of MPV^0, MPV$^+$, MPV^0-GE, and MPV$^+$-GE on HC is 113.3s, 36.1s, 113.8s, and 36.5s, respectively. The reason is that MPV^0 runs the GenAcc algorithm more times to generate a larger-size ADS than MPV$^+$. As for MPV^0-GE, it requires the extra calculations to get accumulative values of digest sets than MPV^0, and hence it costs the most time in ADS generation.

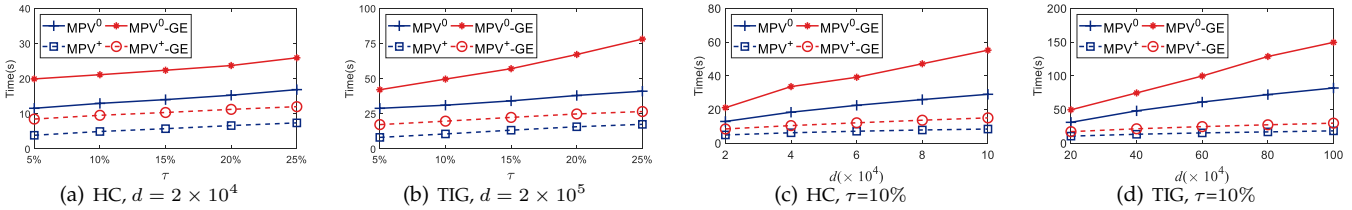
Fig. 7 shows the storage costs on blockchain. From Fig. 7-(a),(b), we can know that the storage costs of all schemes are positively impacted by parameter N. For example, the ADS size of MPV^0 on TIG increases from 97KB to 484KB as N increases from 200 to $1,000$. This is because the more number of keys means the more number of matrices, cubes, or accumulative values need to be generated, hence resulting in larger-size ADSs. For the same reason, the storage costs on TIG are bigger than those on HC. From Fig. 7-(c),(d), we can see that the storage costs of MPV^0 and MPV$^+$ increase as the value of d increases, while MPV^0-GE and MPV$^+$-GE incur constant costs. This is because for a single key, MPV^0 and MPV$^+$ generate a verifiable matrix and a verifiable cube as the ADS, respectively, but the gas-efficient versions generate only a fixed-size accumulative value. Since the sizes of matrix and cube are positively affected by the data size d, the bigger value of d means a larger size of ADSs in both MPV^0 and MPV$^+$. In terms of four schemes, MPV^0 generates the largest size of ADSs, while the gas-efficient versions produce the smallest on-chain ADSs.

VO Construction. As discussed in Section 8.1, the VO construction and verification costs are mainly affected by parameters τ and d, and thus their performance is evaluated with varying values of τ and d. From Fig. 8, we can see that MPV$^+$ and MPV^0-GE consume the least and the most VO construction time under the same settings. For example, when $\tau = 10\%$ and $d = 2 \times 10^4$, the VO construction time of MPV^0, MPV$^+$, MPV^0-GE, and MPV$^+$-GE on HC is 25.8s, 17.5s, 47.8s, and 25.4s, respectively. This is because MPV$^+$ runs algorithm VerProof less times to generate a smaller-size VO, but MPV^0-GE requires generating extra proofs in the two-stage verification compared with MPV^0. From Fig. 8-(a),(b), we know that parameter τ has little influences on the VO construction time of MPV^0 and MPV$^+$. The main



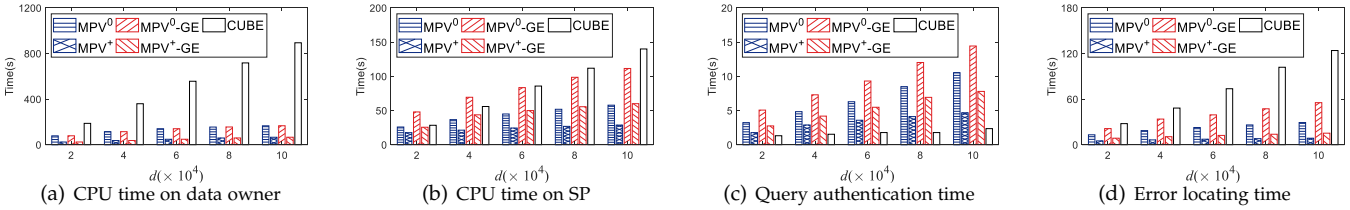
(a) HC, $d = 2 \times 10^4$ (b) TIG, $d = 2 \times 10^5$ (c) HC, τ=10% (d) TIG, τ=10%

Fig. 10: The query authentication time on the user side.



(a) HC, $d = 2 \times 10^4$ (b) TIG, $d = 2 \times 10^5$ (c) HC, τ=10% (d) TIG, τ=10%

Fig. 11: The error locating time on the user side.



(a) CPU time on data owner (b) CPU time on SP (c) Query authentication time (d) Error locating time

Fig. 12: The comparison results between CUBE and our solutions on HC dataset (τ=10%).

reason is that the query results cover all the matrix columns in MPV^0 and all the 2-faces of cube in MPV$^+$ for all the query ratios. From Fig. 8-(c),(d), we can see that the VO construction time of all schemes is positively impacted by parameter d. This is because under a fixed query ratio, the larger d means the more rows or faces covered by the query results, incurring larger VO construction time.

From Fig. 9, we can see the VO size in MPV$^+$ is smaller than that in MPV^0 under the same settings. For example, when $\tau = 10\%$ and $d = 2 \times 10^5$, the VO size of MPV^0 and MPV$^+$ on TIG is 3.8KB and 1.0KB, respectively. This is because the number of faces covered by the query results in MPV$^+$ is less than the number of covered lines and columns in MPV^0. Compared with MPV^0, MPV^0-GE needs to generate a proof for each digest in addition, hence incurring the biggest VO size. Besides, we can see that the VO sizes of all schemes on both datasets are positively impacted by both parameters τ and d. The reason is that when either of τ and d grows, the size of query results $\alpha = \tau \cdot d$ increases, making the number of covered lines/columns or faces increase.

Verification. The verification process consists of query authentication and errors locating. From Fig. 10, we can find that the query authentication time of all schemes increases as the parameters τ and d increase. This is because the bigger τ or d means the more data needed to be verified hence incurring more time. Under the same settings, MPV^0-GE and MPV$^+$ require the most and the least verification time, respectively. For example, when $\tau = 5\%$ and $d = 2 \times 10^5$, the query authentication time of MPV^0, MPV$^+$, MPV^0-GE, and MPV$^+$-GE on TIG is 9.9s, 7.4s, 17.5s, and 9.3s, respectively. This is because MPV$^+$ runs algorithm VerProof less times, but MPV^0-GE requires extra performing digest verification, compared with MPV^0. For the same reason, Fig. 11 shows the similar trends as Fig. 10.

Comparison with Prior Work. As our work is the first attempt to support fine-grained query authentication in the HSB environment, we compare our MPV schemes with a secure cloud storage scheme CUBE [23] that also supports error locating based on three-dimensional cubes and bilinear pairing [36]. The reason for choosing CUBE as the comparative scheme is that both CUBE and our MPV schemes focus on identifying the corrupted data retrieved from an untrusted server by error locating. The main difference is that CUBE does not support range/freshness queries, and verifies outsourced data through a challenge-response phase. For a fair comparison, we assume that the size of challenged data in CUBE is the same as the size of query results in our schemes.

From Fig. 12-(a), we can see that CUBE incurs the most execution time on the data owner side among all schemes. For example, when $d = 4 \times 10^4$, the time executed on the data owner side of MPV^0, MPV$^+$, MPV^0-GE, MPV$^+$-GE, and CUBE is 113.3s, 36.1s, 113.8s, 36.5s and 358.9s, respectively. The main reason is that the bilinear pairing operations as the cryptographic basis of CUBE are more expensive than the RSA accumulator operations used in our schemes. As for the execution time on the SP side shown in Fig. 12-(b), MPV^0, MPV$^+$, and MPV$^+$-GE always perform better than CUBE, and MPV^0-GE performs better than CUBE when the data size d is large enough. For example, the time executed on the SP side of MPV^0-GE and CUBE increases from 47.8s to 83.3s and from 28.3s to 85.6s, respectively, as d increases from 2×10^4 to 6×10^4. This is because CUBE needs to generate a proof for each challenged data and aggregate all of them together, while MPV^0-GE generates a proof for each row and column covered by the query results. Therefore, the data size d has a greater impact on CUBE than MPV^0-GE. From Fig. 12-(c), we can see that the query authentication time of CUBE is the least among all schemes. For example, when $\tau = 10\%$ and $d = 2 \times 10^4$, the query authentication time of MPV^0, MPV$^+$, MPV^0-GE, MPV$^+$-GE and CUBE is 3.2s, 1.8s, 5.1s, 2.7s, and 1.3s, respectively. The reason is that CUBE uses global verification to authenticate the integrity of all challenged results at once. However, CUBE performs the worst among all schemes in the error location phase as illustrated in Fig. 12-(d). For example, under the same

settings ($\tau = 10\%$ and $d = 2\times10^4$), the error locating time of of MPV^0, MPV$^+$, MPV^0-GE, MPV$^+$-GE, and CUBE is 12.9s, 4.9s, 21.1s, 8.5s, and 27.7s, respectively. This is because CUBE needs to perform multiple rounds of verification over a cube-based hierarchical structure to screen out error data.

9 Conclusion

In this paper, we propose two MPV schemes to achieve fine-grained query authentication in the HSB environment. In both schemes, a user not only can effectively verify the data retrieved off-chain, but also can distinguish authentic data from falsified results to gain high data utility. Compared with the basic MPV scheme, the gas-efficient version generates a constant-size on-chain ADS and thus is more suitable for a super-large scale dataset. The empirical study validates the effectiveness of our schemes. As discussed in the article, our MPV schemes support authenticated freshness queries over data streams updated dynamically. As part of our future work, we will try to extend the fine-grained authentication solutions to support other complex queries, such as skyline queries and boolean queries.

Acknowledgments

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, 62372121, U20A20181, 62272162, 62172159, and 61872133, 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. Yu Peng and Mingzuo Xu contributed to the work equally and should be regarded as co-second authors.

References

[1] Y. Miao, Z. Liu, H. Li, K. K. R. Choo, and R. H. Deng, "Privacy-preserving byzantine-robust federated learning via blockchain systems," *IEEE Transactions on Information Forensics and Security*, 2022.

[2] Y. Zhang, K. Gai, J. Xiao, L. Zhu, and K. K. R. Choo, "Blockchain-empowered efficient data sharing in internet of things settings," *IEEE Journal of Selected Areas in Communications*, 2022.

[3] E. Zhou, Z. Hong, Y. Xiao, D. Zhao, Q. Pei, S. Guo, and R. Akerkar, "MSTDB: A hybrid storage-empowered scalable semantic blockchain database," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[4] 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.

[5] B. Zhang, B. Dong, and W. H. Wang, "Integrity authentication for SQL query evaluation on outsourced databases: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[6] C. Xu, C. Zhang, and J. Xu, "vChain: Enabling verifiable boolean range queries over blockchain databases," in *Proc. of SIGMOD*, 2019.

[7] C. Zhang, C. Xu, J. Xu, Y. Tang, and B. Choi, "GEM^2-Tree: A gas-efficient structure for authenticated range queries in blockchain," in *Proc. of ICDE*, 2019.

[8] M. Narasimha, and G. Tsudik, "Authentication of outsourced databases using signature aggregation and chaining," in *Proc. of DASFAA*, 2006.

[9] J. Wang, X. Chen, X. Huang, I. You, and Y. Xiang, "Verifiable auditing for outsourced database in cloud computing," *IEEE Transactions on Computers*, 2015.

[10] Q. Liu, Y. Peng, Q. Xu, H. Jiang, J. Wu, T. Wang, T. Peng, G. Wang, and S. Zhang, "MARS: Enabling verifiable range-aggregate queries in multi-source environments," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[11] Z. Xu, Y. Lin, V. Sandor, Z. Huang, X. Liu, "A lightweight privacy and integrity preserving range query scheme for mobile cloud computing," *Computers & Security*, 2019.

[12] 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.

[13] 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.

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

[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] 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.

[17] H. Jin, K. Zhou, H. Jiang, D. Lei, R. Wei, and C. Li, "Full integrity and freshness for cloud data," *Future Generation Computer Systems*, 2018.

[18] J. Zhu, Q. Li, C. Wang, X. Yuan, Q. Wang and K. Ren, "Enabling generic, verifiable, and secure data search in cloud services," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[19] 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.

[20] Y. Mu, Q. Liu, J. Zhou, K. Xie, and G. Wang, "Achieving flow-oriented reliable services in cloud computing," in *Proc. of ISPA/IUCC*, 2017.

[21] A. Kittur, S. Kauthale and A. Pais, "Bad signature identification in a batch using error detection dodes," in *Proc. of ISEA-ISAP*, 2019.

[22] C.-T. Li, M.-S. Hwang, and S. Chen, "A batch verifying and detecting the illegal signatures," *International Journal of Innovative Computing, Information and Control*, 2010.

[23] G. Xu, Z. Sun, C. Yan, and Y. Gan, "A rapid detection algorithm of corrupted data in cloud storage," *Journal of Parallel and Distributed Computing*, 2018.

[24] H. Yuan, X. Chen, G. Xu, J. Ning, J. K. Liu, and R. H. Deng, "Efficient and verifiable proof of replication with

fast fault localization," in *Proc. of INFOCOM*, 2021.
[25] H. Wang, C. Xu, C. Zhang, J. Xu and Z. Peng, "vChain+: Optimizing verifiable blockchain boolean range queries," in *Proc. of ICDE*, 2022.
[26] X. Dai, J. Xiao, W. Yang, C. Wang, J. Chang, R. Han, and H. Jin, "LVQ: A lightweight verifiable query approach for transaction history in bitcoin," in *Proc. of ICDCS*, 2020.
[27] Y. Peng, M. Du, F. Li, R. Cheng, and D. Song, "FalconDB: Blockchain-based collaborative database," in *Proc. of SIGMOD*, 2020.
[28] M. S. Rahman, I. Khalil, N. Moustafa, A. P. Kalapaaking, and A. Bouras, "A blockchain-enabled privacy-preserving verifiable query framework for securing cloud-assisted industrial internet of things systems," *IEEE Transactions on Industrial Informatics*, 2021.
[29] Y. Zhu, Z. Zhang, C. Jin, A. Zhou, and Y. Yan, "SEBDB: Semantics empowered blockchain database," in *Proc. of ICDE*, 2019.
[30] Q. Pei, E. Zhou, Y. Xiao, D. Zhang, and D. Zhao, "An efficient query scheme for hybrid storage blockchains based on merkle semantic trie," in *Proc. of SRDS*, 2020.
[31] H. Wu, Z. Peng, S. Guo, Y. Yang, and B. Xiao, "VQL: Efficient and verifiable cloud query services for blockchain systems," *IEEE Transactions on Parallel and Distributed Systems*, 2021.
[32] C. Zhang, C. Xu, H. Wang, J. Xu, and B. Choi, "Authenticated keyword search in scalable hybrid-storage blockchains," in *Proc. of ICDE*, 2021.
[33] C. Zhang, Y. Xu, Y. Hu, J. Wu, J. Ren, and Y. Zhang, "A blockchain-based multi-cloud storage data auditing scheme to locate faults," *IEEE Transactions on Cloud Computing*, 2022.
[34] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Proc. of CRYPTO*, 2002.
[35] R. Gennaro, S. Halevi, and T. Rabin, "Secure hash-and-sign signatures without the random oracle," in *Proc. of EUROCRYPT*, 1999.
[36] F. Zhang, R. Safavi-Naini, and W. Susilo, "An efficient signature scheme from bilinear pairings and its applications," in *Proc. of PKC*, 2004.

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

Mingzuo Xu received his B.Sc. in Materials Science and Engineering in 2019 from Hunan University, China. Currently, he is studying for a Master's degree in the College of Computer Science and Electronic Engineering at Hunan University, China. His research interests include cloud computing security and blockchain.

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. 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, network trust and security, and routing protocols. Dr. Wu has regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Services Computing, and Journal of Parallel and Distributed Computing. 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 2011. Currently, he is a professor at the Institute of Artificial Intelligence and Future Networks, Beijing Normal University & UIC, China. His research interests include internet of things and edge computing.

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 his Ph.D. degree in Computer Science, at Central South University, China in 2002. He is a Pearl River Scholarship Distinguished Professor of Higher Education in Guangdong Province, and a Doctoral Supervisor of School of Computer Science and Cyber Engineering, Guangzhou University, China. 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), and blockchain. He is a Distinguished Member of CCF, a Member of IEEE, ACM and IEICE.