IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 11, 2018, accepted January 7, 2019, date of publication January 18, 2019, date of current version February 6, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2892442

Secure Dynamic Big Graph Data: Scalable, Low-Cost Remote Data Integrity Checking

YU LU AND FEI HU, (Member, IEEE)
Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL 35486, USA

Corresponding author: Fei Hu (fei@eng.ua.edu)

This work was supported in part by U.S. NSF DGE#1723250. Any ideas presented here do not necessary represent NSF's opinions.

ABSTRACT As a special type of big data, the big graph has wide applications. The remote data integrity checking (RDIC) scheme enables public clouds to efficiently convince the clients that their graph data are stored properly, without the need of retrieving the actual data contents. The existing schemes support verifiable update often by adopting the authentication data structures (ADSs), e.g., Merkel hash tree (MHT). The main obstacle for applying the existing RDIC schemes to big graphs is due to the lack of support for verifiable sub-graph operations with high efficiency. In this paper, we propose a new RDIC scheme for big graphs, called GVD-RDIC, to support public auditing and verifiable dynamic graph updates with high efficiency. We have designed novel ADS based on the graph Voronoi diagram (GVD) and enhanced MHT to address the integrity of graph structure and verifiable sub-graph updates. In addition, our ADS can be applied to any type of graphs. Moreover, our GVD-RDIC scheme adopts a new construction for the homomorphic authenticator to enable index verifications for public auditors. A server response constructed with unchallenged block will be rejected. The proposed scheme is proven secure under a random oracle model. Both the theoretical analysis and the simulation results show that our scheme is practicable for real-world big graphs.

INDEX TERMS Big data integrity, big graph, public auditing, Internet cloud.

I. INTRODUCTION

The rapid growth in the data volume of real-world graphs, drives the development of distributed, cloud-based big graph computing systems, including Pregel [1], GraphLab [2], Giraph [3], etc. Big graph is a special type of big data. Typical big graph examples include Facebook-like social networks, large wireless/wired networks (with thousands of nodes), geographical terrain maps, big city transportation data, etc. The big graph processing systems often adopt distributed computation models over the shared resources such as storage, computing power, and management services. Due to the elasticity and scalability of distributed computing resources [4] and the pay-as-you-go payment method, adopting cloud systems is considered as an economic solution for real-world graph storage and processing. Clients can enjoy low-cost computing power, and access their graphs anywhere via the Internet. However, a downside is that the data owner loses the direct control of their graphs. Thus, security becomes a major concern. According to a survey from international data corporation enterprise panel, 87% cloud users doubt the security of their data in clouds. Large and complex network systems are vulnerable, and the hackers can manipulate the data or computing process. In some cases even the cloud provider can not be fully trusted. For example, when the cloud storage severs suffer unexpected failures, they may hide the data loss from the data owner to avoid their responsibilities. An investigation [5] shows that 25% of security incidents are related to data loss and leakage.

Due to the large volume of big graphs and relatively constrained computational resources in the clients' machines, it is impracticable for the clients to periodically check whether or not their graphs are stored properly by downloading the entire graph. Thus the critical issue here is to find a more efficient way to check the integrity of the big graph data. Traditional integrity verification methods, such as hash functions and message authentication code (MAC), can not be directly applied here due to the absence of the original data files [6].

As a countermeasure, Ateniese *et al.* [7], [8] introduced provable data possession (PDP) scheme for remote data integrity verification. With PDP the clients can pre-compute some small metadata units called homomorphic authenticators for each data block and sends them to the cloud together with the original dataset. Whenever a client challenges the server for data integrity, the server can efficiently generate a proof based on the datafile and authenticators. The PDP scheme is practicable since it requires only a tiny part of the entire dataset. For example, in Ateniese's paper, they proved that by checking 460 randomly selected blocks among 10,000 blocks the verifier has more than 99% probability to detect 1% data corruptions. Their construction scheme is based on RSA, and makes use of the partial homomorphic property of RSA signature. Shacham and Waters [9] proposed a new construction method called proof of retrievability (POR), which is based on Boneh-Lynn-Shacham (BLS) signature that can reduce the size of homomorphic tags. The BLS signature is based on bilinear map, and the computation cost for paring operations is higher than the signatures based on Elliptic Curve Cryptography (ECC) [10], [11]. These schemes enable efficient integrity checking of static data in the remote servers.

Another important factor to be considered is that the data contents/operations in the clouds are often highly dynamical, since the clients may not only query the clouds but also perform dynamic operations such as updates, insertions and deletions. Thus, it is vital for a RDIC scheme to support scalable, dynamic data operations. Some existing works such as Merkel Hash Tree (MHT) [12] or ranked skipping-list [13], support verifiable dynamic operations by adopting the authentication data structures (ADS) structures.

There are a few challenging issues that we plan to address in this paper. Firstly, no existing work addresses public auditing for dynamic graph data. Let's consider a graph G, where each vertex contains its own content and some structural indicators that tell the relationships among the contents of other vertexes. The contents can include the classification information, indexing results, or the sensitivity of the contents [14]. In [15] the integrity of such relationships is referred as ''structural integrity'', while the integrity of the contents in the vertexes is called ''content integrity''. Therefore, both the content and structure of the updated graph need to be authenticated when verifying dynamic operations. But current schemes only provide content verifications. Although there are some ADS instantiations such as [15]–[18] which may be used to verify the dynamic graph operations, they are either restricted to certain type of graphs or not scalable for dynamic big graph verification operations, especially for subgraph updates.

In this paper we propose a generic, efficient RDIC scheme for dynamic big graphs in clouds. Our contributions include 3 aspects as follows:

1. To the best of our knowledge, this is the first work to fill in the research blank - there is still no secure and general RDIC scheme for big graphs that supports verifiable dynamic graph operations. We achieve the low-cost public auditing and scalable dynamic graph authentication by using a new ADS, namely, graph Voronoi diagram (GVD)-MHT.
2. To ensure that the cloud responds exactly with the block queried, our scheme incorporates with an additional verification process for the indices of block.
3. We provide the detailed proof of the completeness and soundness of our scheme. We have also conducted simulation analysis, and our performance results show that our construction is efficient for real-world big graphs.

This paper is a significant extension of our conference paper [19]. We have focused on a complete, new RDIC scheme in this paper instead of just designing a simple hash scheme as in [19]. Unlike [19], this paper will provide the comprehensive GVD-MHT protocol details. Moreover, it provides the detailed proof process for 3 important theorems in terms of the soundness and completeness of our big graph security scheme.

The rest of paper is organized as follows: Section II introduces the existing works on RDIC system and structure integrity verification. Section III lists the notations and security assumption used in our construction, and reviews the definitions of general RDIC and MHT. Section IV defines the general system and security mode of RDIC scheme for graphs. Section V describes the detailed construction process for the proposed GVD-RDIC, and shows how it authenticates dynamic graph operations. Section VI provides the proof of correctness and soundness for the proposed scheme. Section VII discusses the extensive construction for privacy preserving and efficiency improving. Section VIII analyzes the performance results of our implementation. Section IX concludes this work.

II. RELATED WORKS

Remote integrity checking enables efficient public auditing for outsourced data in clouds. It has attracted many research interests. Starting from the work in [8] there are already much progress in this direction such as [12], [13], and [20]–[24], which explored different construction models to make the auditing schemes more efficient. These works mainly focus on two aspects, i.e., supporting data dynamics and preserving the privacy against the third party auditor (TPA).

A. DATA DYNAMICS

This property enables a data owner to perform dynamic data operations including, but not limited to, insertion, deletion and modification, with acceptable overhead after the graph data is sent to a remote server. In an earlier work [22] a dynamic PDP scheme based on symmetric encryption and hash functions is proposed to support dynamic operation queries. But the number of queries is limited and the insertion is not fully supported. Later on, an extended PDP scheme is proposed in [23] by adopting a modified, ranked skipping-list to support fully dynamic data operations. The basic idea is to replace the indices of data blocks in the homomorphic authenticator with the hash values of data blocks, and use an ADS to authenticate the correctness of the hashing operations. While the data may be modified, only the hash and authenticator of the modified block need to be updated. Although in this scheme public audit is not supported and the size of data block is fixed, the idea inspires more promising solutions. For example, a MHT-based scheme [13] is used to support fully dynamic data operations. Such a work was enhanced by [21] to support fine-grained dynamic operations.

These works improved the scalability of RDIC scheme on dynamic data. However, replacing the indices of data blocks makes the scheme not sound in default auditing process. Because the public auditor does not have the original data blocks, without indices it cannot verify whether or not the returned authenticator is for the exact challenged data block. Thus these schemes rely on the assumption that the server will be honest during auditing, which contradicts the typical assumption that the server may not be honest. Some other works attempt to make ADS-based RDIC scheme more efficient in different scenarios such as multi-replication environment [25], [26]. In [26], two indices are used incorporate with the original MHT to precisely indicate the level of a node v in MHT and maximum number of leaf nodes that can be reached from v. While auditing, TPA can computer the exact index of a block to make sure the server do not response with another block on the verification path.

B. PRIVACY AGAINST THIRD PARTY AUDITOR

While adopting public auditor brings significant benefits, it also raises new security issues. For example, the TPA may gain enough information to retrieval the original data through multiple challenges. This problem is essential when the data is sensitive. Normally, data confidentiality can be achieved by using symmetric encryption scheme to encrypt the dataset. But in cloud computing applications, the encryption will increase unnecessary overhead. Hence, in [20] a random mask based scheme is proposed to address this problem. Some recent works [6], [13] zero-knowledge proof is introduced to preserve the data privacy. The process of identity-based RDIC was defined in [6] to reduce the complexity of key management. Some other works [27], [28] on medical data access control also adopts zero-knowledge proof to audit whether the file is encapsulated with correct access rule. The goal of construction is quite different from RDIC scheme, but they can also achieve data privacy through zero-knowledge proof.

Another aspect of this research is related to ADS for dynamic graph data. Some existing works such as [14]–[17], have made much progress on this issue. The original MHT [15] can authenticate tree-structured graphs. Later on a scheme called search-DAG [16] was proposed to enable authentication for both trees and DAGs. An efficient scheme for graphs with different structures was proposed in [14] by introducing graph traversal process. But the weights (contents) in the edges are not authenticated. Thus it does not work for weighted graph, a follow-up work [17] improved the privacy. But they are not efficient enough when authenticating dynamic graph operations, for an insertion or deletion operation, the complexity can be O(N) [17], here N is the size of the whole graph. Thus the applications are restricted to the scenarios that the graph data has a specific type and do not change frequently, which contradicts the dynamic nature of big-graph applications.

III. PRELIMINARIES

Notations: A graph $G = (V, E)$ contains two sets: V denotes all vertexes in the graph and E contains all the edges of the graph. An edge $e(u, v, w)$ in E is an edge from vertex u to v with the weight w. For an undirected graph, an edge can be represented as two directed edges. In a graph the ***source nodes*** mean the nodes without incoming edges. For a vertex v, if another vertex u is on the path from v to a source node, then v is a ***posterity*** of u. The ***identical graphs*** are the graphs with exactly the same content and structure.

Bilinear Map: Our constructions make use of symmetric bilinear prime order groups $\mathbb{G}_r = (e, \mathbb{G}_1, \mathbb{G}_2, p, g)$ with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Here $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of order p; and g are the generators of $\mathbb{G}_1$. The map e is bilinear if it has the following three properties:

Bilinearity: $\forall g_a, g_b \in G_1, e(g_a^n, g_b^m) = e(g_a, g_b)^{mn}$;

None-degeneracy: $\forall g \neq 0 \in \mathbb{G}_1, e(g, g) \neq 1$;

Computational Efficiency: $\forall g_a, g_b \in \mathbb{G}_1, e(g_a, g_b)$ can be computed in polynomial time.

Also, two hash functions $\mathsf{H}_1 : (0, 1)^* \rightarrow \mathbb{G}_1, \mathsf{H}_2 : (0, 1)^* \rightarrow (0, 1)^l$ are required; here H_1 is a map to point hash function which can map a message to an element in $\mathbb{G}_1$, and H_2 is a cryptographic hash function that maps a message to a fix-size hash value. Under these setups our construction can be proven secure under the following assumption:

Assumption 1 (Computational Diffie-Hellman): The Computational Diffie-Hellman (CDH) assumption holds if every probabilistic-polynomial-time (p.p.t) adversary $\mathcal{A}$ can solve the CDH problem (For the cyclic group $\mathbb{G}_1$ of prime order p and $a, b \overset{s}{\leftarrow} \mathbb{Z}_q$ here q is a prime number. Given g, g^a and g^b, compute g^{ab} without knowing a, b.) only with a negligible probability.

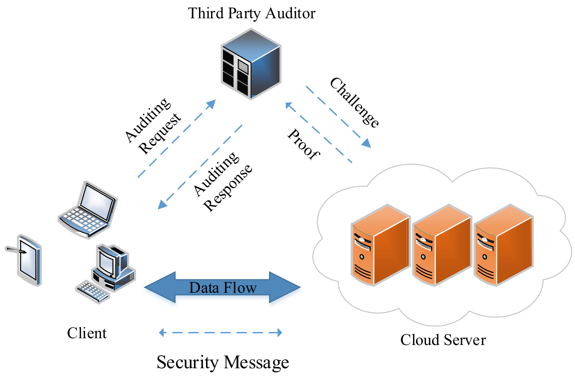

FIGURE 1. The system model of general RDIC.

A. GENERAL RDIC SCHEME

In a public auditing scheme (Fig.1), there are three entities:

Client (C): Data owner or data user, who can request a cloud provider for efficient computation and storage. It has fully authority to its part of datasets.

Cloud Server (CS): Remote data sever managed by a cloud provider, which has the required storage and computing resources.

Third Party Auditor (TPA): An entity authorized by a client to periodically check the integrity of the data stored on cloud.

Here each entity has its own capability and interests. The client has a big graph stored in the cloud without a local copy, and wants to ensure the security of its graph with the minimum cost. The cloud server has large storage space and computation power to provide the service to the client, but it has its own interest such as keeping good reputation. Thus it may try to hide the possible data corruptions from the user. However, we assume that the CS will not deliberately reveal the data to the third party or manipulate data because there is no benefit for doing that. TPA is capable of periodically running the auditing process and is trusted by the clients, and may have interest in retrieving the sensitive information from the clients data through the auditing process.

A public-key-based RDIC scheme $IC = (Setup,\ KeyGen,\ TagGen,\ Challenge,\ Proof,\ Verify)$ consists of six algorithm as described below:

- $pp \leftarrow Setup(1^\lambda)$: C takes the security parameters as inputs, and outputs the public parameter pp.
- $sk, pk \leftarrow KeyGen(pp)$: C takes the public parameter pp and generates a public key pair (sk, pk).
- $signauth, \sigma \leftarrow TagGen(pp, sk, F)$: C takes the secret key sk and file block F, returns the signature $signauth$ and a homomorphic authenticator σ for the block.
- $c \leftarrow Challenge(pp, signauth)$: TPA takes pp and signature $signauth$, and generates a challenge c.
- $\pi \leftarrow Proof(pp, F, \sigma, signauth, c)$: CS takes pp, σ,F and $signauth$ to generate a proof π for the challenge c.
- $b \leftarrow Verify(pk, \pi)$: TPA takes the public key pk to verify the returned proof π, and outputs either $b = 1$ (means Accept) or $b = 0$ (Reject).

A RDIC system has three major security concerns: completeness, soundness, and zero knowledge privacy against the TPA. Here we mainly focus on completeness and soundness, and privacy issue will be discussed in Section VII.

Definition 2 (Completeness): A RDIC scheme IC is complete if for any valid server P and a proof π from P, we have:

$$Pr\,[Reject \leftarrow Verify(sk, \pi, c)] \approx 0$$

Definition 3 (Soundness): A RDIC scheme IC is sound if, for a valid data block m and any data block m' ($m' \neq m$) and for all p.p.t attacker $\mathcal{A}$, we have:

$$Pr\begin{bmatrix} (pk, sk) \leftarrow \mathcal{A}^{KeyGen}(1^{pp}) \\ \pi \leftarrow \mathcal{A}^{Proof}(pk, c, m', \sigma)) \\ (b = 1) \leftarrow Verify(pk, \pi, c) \end{bmatrix} \approx 0$$

B. MERKLE HASH TREE(MHT)

MHT [15] is a widely used cryptographic technology for large-scale database hash management. The basic idea of MHT is aggregating the hashes of all data units in a dataset to one or multiple root hashes following some data structure. In this way, MHT can support efficient verification for dataset changes. Here we illustrate the basic procedure of MHT construction with a simple Binary hash tree (BHT) case. The whole datasets are handled as independent data unions m_i. The leaf level of BHT has the hashes of all data blocks, and each non-leaf-level node ($LN = H_2(m_i)$) contains the hash of the concatenation of all the hashes in all its children nodes (in BHT each node has two children, thus $NLN = H_2(H_l||H_r)$), Through bottom-up operations it will generate a root hash representing the whole dataset.

IV. RDIC MODEL AND SECURITY MODEL

In this section we describe the system model of RDIC scheme for big graphs and its security model.

A. RDIC SYSTEM FOR BIG GRAPHS

Our goal is to design a RDIC scheme for big graphs and ensure that it supports dynamic updates. Recall that we are interested in a scheme that is able to authenticate both content integrity and structure integrity. Naturally there are two ways to achieve this: either including structure information in data unions or using structure information in signature aggregations. Either way the structure information of the graph is required. Therefore, a RDIC scheme for graph IC_G consists of seven algorithms IC_G=*(Setup, Extract, KeyGen, Sigag, TagGen, ChallengeGen, ProveGen, Verify)*:

- $pp \leftarrow Setup(1^\lambda)$: C takes inputs the security parameter, outputs the public parameter pp.
- $G_t \leftarrow StrGain(G, set)$: C takes the graph G, generates a labeled graph G_t that can indicate the structure of graph G.
- $sk, pk \leftarrow KeyGen(pp)$: C takes the public parameter pp, and generates a public key pair (sk, pk).
- $signauth \leftarrow Sigag(pp, G_l, sk)$: C takes the secret key sk and the labeled graph G_l, computes the hash value h_i for each storage unit, then constructs the corresponding ADS and generates a root hash R for the whole graph. It returns the signature $signauth$ of R, signed by the secret key sk.
- $\sigma, \tau \leftarrow TagGen(pp, sk, G_l)$: C takes the secret key sk and the graph G_l, returns the homomorphic authenticators σ for all storage blocks in the graph together with a verification package τ that contains the required integrity checking information for TPA.
- $chall \leftarrow Challenge(pp, signauth)$: TPA takes pp and signature $signauth$, generates a challenge $chal$.
- $\pi \leftarrow Proof(pp, G_l, \sigma, signauth, chal)$: CS takes pp, σ, G_l and $signauth$ to generate a proof π for the challenge $chal$.
- $b \leftarrow Verify(pk, \pi)$: TPA takes the public key PK to verify the returned proof π, and either outputs $b = 1$ (Accept) or $b = 0$ (Reject).

B. SECURITY MODEL

This work mainly considers two security properties: completeness and soundness. Another important aspect is data privacy. But it is not the main focus of this study since we want to emphasize our contributions in dynamic graph operation integrity issues. We will briefly discuss privacy issue in the discussion section (Section VII) since some existing works have well addressed this problem. Completeness stands if the verifier always accepts the proof from a valid server.

Definition 4: A RDIC scheme for graphs is complete, if for all key pairs (sk, pk) generated by *KeyGen*, all graphs G and the tags (σ, τ) from $TagGen(sk, G_l)$, *Verify* will always output Accept, i.e.,

$$(Proof\ (G_l, \tau) \leftrightarrow Verify(pk, \tau)) = 1$$

Following the precise definition of soundness from Shacham and Waters's work [9], we define the soundness of a RDIC scheme for graphs that supports dynamic graph operations as follows.

Definition 5: A RDIC scheme for graphs is $\epsilon-$sound if there exists an efficient extraction algorithm *Extr*, such that for every *p.p.t* adversary $\mathcal{A}$ who wins the forge game (see below) and generates a $\epsilon-$admissible cheating prover P' for a graph G, *Extr* is always able to recover G, i.e., $Extr(sk, pk, pp, signauth, \tau, P')$, with a negligible probability.

Forge Game

- $sk', pk \leftarrow \mathcal{A}^{KeyGen}(pp)$: $\mathcal{A}$ can generate any secret key $sk' \neq sk$; but TPA will still use the public key from C.
- $signauth' \leftarrow \mathcal{A}^{Sigag}(pp, G_t', sk')$: $\mathcal{A}$ can undertake the execution *Sigag* with any graph and secret key.
- $\sigma', \tau \leftarrow \mathcal{A}^{TagGen}(pp, sk', G_t')$: $\mathcal{A}$ can query the TagGen oracle with any graph and a specific secret key, and C generates its secret key sk and computes tags with graph G, then returns the set of tags to $\mathcal{A}$ together with the verification package τ.
- $\pi' \leftarrow \mathcal{A}^{Proof}(pp, \sigma', signauth', chal)$: $\mathcal{A}$ can execute *Proof* for a graph G it queried in *TagGen* phase. It can specify the secret key sk' and signature $signauth'$ to behave as a prover, and it can also get the output from the verifier.
- $\mathcal{A}$ wins if $b = 1 \leftarrow Verify(pk, \pi')$.

V. PROPOSED CONSTRUCTION SCHEME

In this section we present our proposed graph Voronoi diagram based PDA scheme (GVD-RDIC). To achieve public auditability, a PKI-based (e.g., BLS short signature or RSA signature) homomorphic authenticator is required. In the following description, we use the notion of BLS signature to illustrate our scheme. It can also be implemented with RSA signature. As mentioned before, the existing ADS-based RDIC schemes have a critical soundness problem, that is, the TPA can not distinguish between two valid proofs of two sets of data blocks, since they remove the file index information in the authenticator. According to the system model, CS can not be fully trusted, since it may try to hide the data corruption. And CS may falsify a valid proof for a corrupted data block. Thus the scheme is not sound. Our scheme fixes this problem by modifying the MHT structure and homomorphic authenticator. Here is our concrete construction process:

$pp \leftarrow Setup(1^\lambda)$: C takes input the security parameter, outputs the public parameter $pp = \{\mathbb{G}_r, \mathsf{H}_1, \mathsf{H}_2\}$.

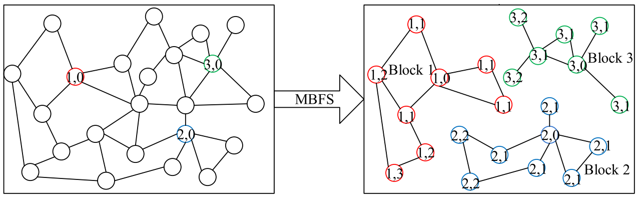

FIGURE 2. Voronoi blocks and labels of connected components (CCs) through MBFS.

$G_t \leftarrow GVD\_StrGain(G, ini)$: Here we use the GVD partitioner [18] approach which uses multi-source breadth first search (MBFS) to extract structural information from graph G with initial parameters $ini = \{S, b_{max}, d_{max}\}$, here the parameter S is the set of all source vertexes of each Voronoia block defined by C. b_{max} and d_{max} are two parameters used to limit the size of Voronoi blocks, b_{max} indicates the largest number of vertexes in one block and d_{max} indicates the highest depth of a block. Also, here we assume the ID of all vertexes in the graph are mapped to a set with arithmetic progression of length one (this is a common operation in graph dataset and does not change the contents of the graph). As an example, Fig.2 shows how *GVD_StrGain* works on a connected component (CC) with 24 vertexes. The basic procedure is as follows:

1. Label each source vertex v_s in the source list S as $(vb(v_s) = i)||bl(v_s) = 0||ID(v_s)||content)$. Here $vb(v)$ is the ID of the Voronoi block for a source vertex. It is a sequence in the set S. Here $bl(v)$ indicates the level of a vertex in an Voronoi block; for the source vertex the number is 0. Starting from the source vertexes, they broadcast $vb(v)$ and bl(v) to the neighbors. For a non-source vertex, when it receives $vb(v_{parent})$ from one of its parent vertexes in the first time, it will assign $vb(v) = vb(v_{parent})$, and $bl(v) = bl(v_{parent}) + 1$. If several broadcast messages arrived at the same time, it will randomly select one of them.
2. After all vertexes vote to halt (i.e., no vertexes in the next round of broadcast or the block level exceeds the limit), some small CCs may remain unassigned, and some Voronoi blocks may exceed the size limit. For huge Voronoi blocks, we break it into several small ones; and for unassigned vertexes, we run BFS starting from the vertexes with a smaller ID until all vertexes are labeled. While all vertexes are labeled, we can output the labeled graph G_l. In practice, *GVD_StrGain* can be executed under the distributed graph computing model [18] with MBFS and hash-min to improve the efficiency.
3. $sk, pk \leftarrow KeyGen(1^\lambda, pp)$: C takes the public parameter pp and chooses a random secret value $\alpha \overset{s}{\leftarrow} \mathbb{Z}_p^*$ as the secret key and computes $v = g^\alpha$. Here the set (v, g) is the public key. Thus, C outputs $sk = \alpha, pk = (v, g)$, keeps sk and sends pk to TPA and CS.

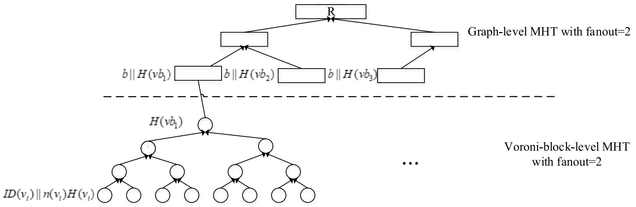

FIGURE 3. Two-level VB-MHT for a big graph.

4. *signauth* ← *Sigag*(pp, G_l, sk): C takes the secret key sk, public parameter pp, and the graph G_t to construct a two-level (Voronoi block) VB_MHT, as shown in Fig. 3. We assume that each vertex (including contents, labels and out-edges) is divided into n storage blocks (potentially encoded by using Reed-Solomon codes [29]); $v_{i,1}, v_{i,2}, \ldots, v_{i,n} \in \mathbb{Z}_q$, here q is a large prime number. For the bottom level (Voronoi block level MHT), the hash value in each leaf vertex is the hash value of all data blocks belonging to a vertex. In order to make valid proofs of two different data blocks distinguishable by TPA, we introduce two indices $(ID(v_i), n(v_i))$. For each leaf node, here ID(v) is the ID of this vertex and n is the number of data blocks under this vertex. Thus, the value in the leaf node will be $(ID(v_i)||n(v_i)||H_2(v_{i,1}|| \cdots ||v_{i,n}))$. For all none-leaf node the value will be $H_2(H(left\_sbling)||Right\_sbling)$. Then, it works bottom up to get a root hash for each Voronoi block as $H_2(vb)$. For the top level MHT, the leaf-node will be the root hash of each Voronoi block with an indicator which tells that it is a root hash of Voronoi block as $b||H_2(vb)$. Then it merges up to get the root hash of the top-level MHT, denoted as R, which will be signed by secret key Îś as $(H_2(R))^{\alpha}$ as the signature. The corresponding ADS then generates a root hash R for the whole graph. Finally, it returns the signature *signauth* of R, signed by the secrete key sk.
5. Φ, τ ← *TagGen*$(pp, sk, G_l, signauth)$: C takes the secret key sk and the labeled graph G_l. For all blocks $v_{i,j}$ here $i \in m$ and $j \in n(v_i)$, the client first generates an auditor package τ for the graph as $\tau = max(n(v_i))||m||u||signauth$. Here $u \xleftarrow{S} \mathbb{G}_1$ is a random selected element from group $\mathbb{G}_1$. Here m is the total number of vertexes in the graph and $max(n(v_i))$ is the number of data blocks in the largest vertex. Then C computes the set of authenticators $\Phi = \{\sigma_{i,j}\}$ for each block $v_{i,j}$ as $\sigma_{i,j} = (H_1(H_2(v_i||j)u^{v_{i,j}})^{\alpha}$. C will send $\{G_l, \Phi, \tau\}$ to CS and τ to TPA. Then it deletes them from the local storage.
6. *chal* ← *Challenge*$(pp, signauth)$: To generate a challenge for checking the integrity of graph, TPA first selects a random $c - element$ subset $I \xleftarrow{S} \{[1, m], [1, max(n(v_i))]\}$. Then for each $I_{i,j} \in I$, TPA chooses a random element $x_{i,j} \in \mathbb{Z}_p^*$. The challenge request sent to CS will be $chal = \{(i, j, x_{i,j}), signauth\}$.
7. π ← *Proof*$(pp, pk, G_l, \sigma, signauth, chal)$: CS first checks the signature *signauth*. If it is invalid, it rejects the request; otherwise, it computes $\mu = \sum_{i,j} v_{i,(j \bmod n(v_i))} x_{i,j} \in \mathbb{Z}_q$ and $\sigma = \prod_{i,j\in I} \sigma_{i,(j \bmod n(v_i))}^{x_{i,j}} \in \mathbb{G}_1$. And the CS will retrieve the leaf hash for the requested vertexes and auxiliary information(AAI) Ω_i of the MHT used for computing the root. Therefore, the proof $\pi = \{\mu, \sigma, (ID(v_i)||n(v_i)||H_2(v_i), \Omega_i\}$ will be returned to the TPA.
8. b ← *Verify*$(pk, \pi, signauth)$: TPA takes the public key PK to verify the returned proof π. First, it computes the root R based on $(ID(v_i)||n(v_i)||H_2(v_i)$ and verifies it with *signauth*. If it is an invalid output $b = 0$ it rejects it; otherwise, it computes $e(\sigma, g) \overset{?}{=} e(\prod_{i,j\in I} H_1(H_2(v_i||j \bmod n(v_i))^{x_{i,j}} \cdot u^{\mu}, v)$. If not matching, the output is $b = 0$; else, the output is $b = 1$ (i.e., accept the proof).

Steps 6 8 belong to **basic integrity verification process**. The whole procedure is demonstrated in Fig. 4.

A. DYNAMIC GRAPH OPERATIONS WITH VB-MHT

In this section, we will show how our scheme supports dynamic graph operations with integrity assurance. Generally, there are three types of graph operations, i.e., *Update*(U), *Insertion*(I), *Deletion*(D). In our scheme, the verification process performs better than the existing schemes since we have considered the case that in some operations the traversal labels in vertexes and rank indices in MHT need to be updated. Our scheme is computationally efficient since all operations are isolated in its own Voronoi blocks. The basic procedure is shown in Fig. 5.

1) UPDATE

In a graph, the update means the change of the content in any vertex or the weight of any edge. Thus the structure of the graph remains the same, and the labels in vertexes remain unchanged. Upon receiving an update request $OP = \{U, v_i', (\sigma_i'), sigauth\}$, CS first checks the signature *sigauth* to determine whether or not the request is from C. If not, it rejects the request; otherwise, it updates $\sigma_{i,j}$ to $\sigma_{i,j}'$ and v_i to v_i', and then replaces $(ID(v_i)||n(v_i)||H_2(v_i))$ with $(ID(v_i)||n(v_i')||H_2(v_i'))$, and retrieves the AAI Ω_i to compute the new root of MHT as R'. CS generates a proof $P_U = (R', \Omega_i, ID(v_i)||n(v_i)||H_2(v_i))$ and sends it to C. After receiving the response, the client first checks the correctness of Ω_i by verifying *sigauth* with $\Omega_i, ID(v_i)||n(v_i)||H_2(v_i)$ and sk. If it fails, it rejects the update; otherwise, it computes and verifies the new root R'. If fails, reject the update; else, update the *sigauth* to $(R')^{\alpha}$. Since the update may also change $max(n(v_i))$, it updates τ with new $max(n(v_i))$ and *sigauth*, and then sends it to TPA and CS.

2) INSERTION

Insertion is more complicated since some insertion may change the structure of graph, and thus makes the structural labels in vertexes out of date. Assume the cloud receives an Insertion request from the client as

TPA **CS**

1.Randomly select a challenge set $c = \{(i,j), x_{i,j}\}_{(i,j)\in I}$;

2.Generate a challenge $chal = (c, sigauth)$;

$\xrightarrow{chal}$

3.Veirfy $sigauth,$ if not valid then abort the process;

4.Retrive and compute:

$$\mu = \sum_{i,j\in I} v_{i,j} \mod n(v_i) \cdot x_{i,j}$$

$$\sigma = \prod_{i,j\in I} \sigma_{i,j}^{x_{i,j}} \mod n(v_i)$$

$$O = ID(v_i)||n(v_i)||H_3(v_i)_{(i,j)\in I}$$

$$\Omega = \{\Omega_i\}_{(i,j\in I)}$$

$\xleftarrow{\{\mu, \sigma, O, \Omega\}}$

6.Compute and verify Root$Rwith(O, \Omega)$.

Reject the proof if fail.

7.Compute and verify

$$e(\sigma, g) \overset{?}{=} e(\prod_{i,j\in I} H_1(H_2(v_i||j \mod n(v_i))^{x_{i,j}} \cdot u^{\mu}, v)$$

Reject the proof if fail, otherwise accept.

FIGURE 4. Protocol for defalut remote graph integrity checking.

C **CS**

1.Generate an operation request:

$OP = \{(Type, \{v_i^{'}\}, \{\sigma_{i,j}^{'}\}, sigauth\};$

$\xrightarrow{OP}$

2.Veirfy $sigauth$

If not valid then abort the process;

3.Update G, σ and compute:

New root of GVD-MHT $R^{'}$

$hash - AAI = (ID(v_i^{'})||n(v_i^{'})||H_3(v_i^{'}), \Omega_i)$

$\xleftarrow{\{R^{'}, hash - AAI\}}$

4.Computer and verify $\Omega_i witholdMHT$

If not valid then abort the process;

5.Compute $R_{new}$with$hash - AAI$

Verify the dynamic opertion by checking$R_{new} \overset{?}{=} R^{'}$

Abort the proof if fail.

6.Sign a new signature$sigauth^{'}$ using$R^{‘}$

Send$sigauth^{'}$ to CS and TPA

FIGURE 5. Protocol for verifiable dynamic graph operations.

$OP = \{I, v_i', (\sigma_i'), sigauth\}$. For an inserted new vertex v_{new} or new edge $e_{new} = (v_s, v_d, content)$, there are three types of insertion operations, as shown in Fig. 6:

1. If v_{new} or v_s do not exist in the graph then a new Voronoi block will be created and the tags of all blocks belonging to this vertex will be computed.

2. If v_s and v_d belongs to one block and v_s $(b(v_s) = b(v_d))$ is two level higher than $(bl(v_s) - bl(v_d) > 1)$, then for all the posterities of v_d in the block their traversal labels needs to be updated thus the corresponding hash value and authentication tags same as performing update operation.
3. Otherwise, the insertion will only affect one vertex.

3) DELETION

For a deletion operation, we have $OP = \{D, v_i', (\sigma_i'), sigauth\}$. In order to avoid frequent change of the graph structure, we introduce an indicator ϱ for each Voronoi block. It will temporarily set the deleted vertexes and edges as deleted form, but not exactly in deleted status as $v = (b(v)|| bl(v) = -1||ID(v))$ or $e = (v_s, v_d, -1)$.

VI. SECURITY ANALYSIS OF THE NEW SCHEME

In this section, we analyze the security of our scheme based on the definitions in Section IV .

A. COMPLETENESS

Theorem 6: If the server and data owner are honest then the valid proof for any random challenge will pass the verification.

Proof: Set $j \mod n(v_i)$ to j'

$$\begin{aligned} e(\sigma, g) &= e(\prod_{(i,j)\in I} \sigma_{i,j'}^{x_{i,j}}, g) \\ &= e(\prod_{(i,j)\in I} (H_1(H_2(v_i)||j' u^{v_{i,j'}})^{\alpha \cdot x_{i,j}}, g) \\ &= e(\prod_{(i,j)\in I} (H_1(H_2(v_i)||j') \cdot u^{\sum_{(i,j)\in I} v_{i,j'}})^{x_{i,j}}, g^{\alpha}) \\ &= e(\prod_{i,j\in I} H_1(H_2(v_i)||j')^{x_{i,j}} \cdot u^{\mu}, v) \end{aligned}$$

B. SOUNDNESS

Theorem 7: Given the root R of VB-MHT and the ID of a vertex v_i. If the hash function H_2 is collision-resistant, then in random oracle no p.p.t adversarial $\mathcal{A}$ can convince the verifier to accept an invalid hash-AAI tuple with non-negligible probability, except for the tuple is computed with the exact vertex queried.

Proof: $\mathcal{A}$ has access to the VB-MHT and the random oracle H_2. For any query to H_2 it will output a random bit-string; For a vertex v_i in G, the valid hash tuple is $ID(v_i)||n(v_i)||H_2(v_i)$, Ω_i. Based on these settings, $\mathcal{A}$ can possibly generate an invalid hash-AAI tuple that passes the verification in the following ways:

1. $\mathcal{A}$ generates the hash-AAI with vertex v which does not exist in the graph. This can be achieved through two different ways: 1) find $H_2(v_i) = H_2(v)$, keep other parts unchanged. However it contradicts the assumption that H_2 is collision-resistant; 2) compute $H_2(v_i) \neq H_2(v')$ and $\Omega_i' \neq \Omega_i$, but merge up to R. In MHT any none-leaf node is computed as $H_2(hash_{leftchild}||hash_{rightchild})$, assuming the node on the path to the root in the valid VB-MHT is $vh = \{h_1, h_2, \ldots, h_n, R\}$, and for the forged VB-MHT the path is $vh' = \{h_1', h_2', .., h_m', R\}$. Then at least one node in vh' is the same as one node in vh, but with different $(hash_{leftchild}||hash_{rightchild})' \neq (hash_{leftchild}||hash_{rightchild})$. This indicates that $\mathcal{A}$ finds a collision in H_2, which contradicts the assumption;
2. Another possible way is that $\mathcal{A}$ uses existing nodes in MHT to forge a hash-AAI. If a non-leaf node is chosen, since the verifier will verify the $ID(v_i)$ first and all non-leaf nodes do not contain $ID(v_i)$, the authentication will fail. If another leaf node is chosen, since the verifier knows the $ID(v_i)$, it will retrieve and check $ID(v_j)$. Since $ID(v_j) \neq ID(v_i)$, the authentication will fail.

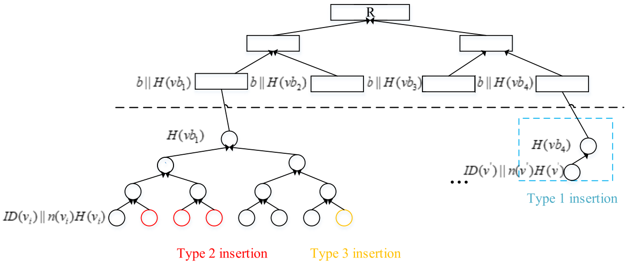

FIGURE 6. Simple examples of three type insertions.

We can thus conclude that the verifier will not accept any invalid hash-AAI with non-negligible probability, unless it is computed with the exact vertex queried.

Theorem 8: If the Computational Diffie-Hellman is hard in bilinear group, the two hash functions (H_1, H_2) are collision-resistant and the signature scheme used is existentially unforgeable, then in the random oracle model, no p.p.t adversarial can convince the verifier to accept an invalid proof with non-negligible probability.

Proof: First the two hash functions (h1, h2) used in our scheme can be considered as collision-resistant, since H_2 is a standard cryptograph hash function proven to be collision-resistant and H_1 is the map to point hash function used in BLS that is also proven collision-resistant in [29]. We can thus say TPA can verify whether the server returns the correct hash-AAI tuple.

Then we can prove the soundness of our scheme with a sequence of games similar to [9]. Start from the forge games defined in Section IV (denoted as Game 0).

Game 1: It is similar to Game 0. In this game the challenger will keep a list of tags with store-query. If the tag from prover has valid signature but not in the list, then the challenger declares failure and aborts.

Based on the analysis from [9], if an adversary causes the challenger aborts in Game1, the adversary is able to construct a forger. And for Game 1, during the verification process only the valid tags generated and signed by the challenger will be used. Game 2 is similar to Game 3. Only the challenger keeps a list of the responses to TagGen queries made by adversary. Then the challenger observes the responses from the prover P'. If any response passes the verification process except the adversary's aggregated tag $\sigma \neq \prod_{(i,j)\in I} \sigma_{i,j}^{x_{i,j}} \bmod n(v_i)$, the challenger declares failure and aborts.

Analysis: If the response passes the verification, then

$$e(\sigma, g) = (\prod_{i,j\in I} H_1(H_2(v_i)\|j \mod n(v_i))^{x_{i,j}} \cdot u^{\mu}, v)$$

However, in this Game the challenger aborts. Thus $\sigma \neq \sigma'$, but it still passes the verification, i.e,

$$e(\sigma', g) = (\prod_{i,j\in I} H_1(H_2(v_i)\|j \mod n(v_i))^{x_{i,j}} \cdot u^{\mu'}, v)$$

In the above equation the correctness of $H_2(v_i)\|j$ mod $n(v_i)$ can be verified through hash-AAI tuple as proved in Theorem 6. Thus if $\mu'_{i,j} = \mu_{i,j}$ for all i, j, we can get $\sigma' = \sigma$ which contradicts the assumption. Therefore for $\Delta\mu_{i,j} \overset{def}{=} \mu'_{i,j} - \mu_{i,j}$ there must exist at least one non-zero element in ΔÎij$_{i,j}$.

We now show whether the adversary causes Game 2 to abort with nonnegligible probability. Then a simulator can be constructed to solve the CDH problem.

The given input of the simulator is $g, g^{alpha}, h \in \mathbb{G}_1$. The goal is to compute h^{alpha}. The simulator works like the challenger in Game 1 but with the following differences:

- The simulator programs the random oracle H_1. Whenever it answers $\mathcal{A}$'s queries, it responds with an element $g^r \in \mathbb{G}_1$, here $r \xrightarrow{s} \mathbb{Z}_p$.
- When the simulator is asked to store a graph G with n vertexes and each vertex consists of m blocks as the set $v_{i,j}$, since the hash function H_2 is collision-resistant and each vertex is different, $H_2(v_i)$ is unique for each v_i. If the block $v_(i, j)$ has not been queried before, $H_1(H_2(v_i)\|j)$ can be generated as follows:
  For each block $v_{i,j}$, the simulator randomly chooses $r_{i,j}, \beta, \gamma \xrightarrow{s} \mathbb{Z}_p$. We have an element $u_{i,j} = g^{\beta}h^{\gamma} \in \mathbb{G}_1$.We have $H_1(H_2(v_i)\|j) = g^{r_{i,j}}/g^{\beta\cdot v_{i,j}}h^{\gamma\cdot v_{i,j}}$ then it computes the tag as: $\sigma_{i,j} = (H_1(H_2(v_i)\|j) \cdot u^{v_{i,j}})^{\alpha} = (g^{\beta\cdot v_{i,j}}h^{\gamma\cdot v_{i,j}}g^{r_{i,j}}/g^{\beta\cdot v_{i,j}}h^{\gamma\cdot v_{i,j}})^{\alpha} = (g^{\alpha})^{r_{i,j}}$
- When the condition in Game 2 occurs, by dividing the forged signature σ' with the expected signature σ in the verification process, we get $e(\sigma'/\sigma, g) = e(\prod_{i,j\in I} u^{\Delta\mu_{i,j}}, v) = e(\prod_{i,j\in I}(g^{\beta}h^{\gamma})^{\Delta\mu_{i,j}}, v)$. Now we can deduce the solution of CDH problem as:

$$h^{\alpha} = (\sigma' \cdot \sigma^{-1} \cdot v^{-\beta\sum_{i,j\in I}\Delta\mu_{i,j}})^{\frac{1}{\gamma\sum_{i,j\in I}\Delta\mu_{i,j}}}$$

As we assumed in Game 2, not all elements in $\Delta\mu_{i,j}$ are zero. Thus the exponent will not cause a division by zero. Also γ is hidden from $\mathcal{A}$ (only u is sent to the adversary). Thus the probability of adversary winning the game is the probability of $\gamma\sum_{i,j\in I}\Delta\mu_{i,j} = 0 \mod p$, which is $1/p$ and can be considered as negligible.

Game 3 is similar to Game 2, except that in this game the challenger aborts if the adversary is successful. But at least one $v_{i,j}$ is not equal to the expected data block in $\sum_{i,j\in I} v_{i,j} \cdot x_{i,j}$.

Analysis: In Game 3 we know $\sigma' = \sigma$ and the $H_1(H_2(v_i)\|j)$ remains the same. Thus we have the verification equation as:

$$e(\prod_{i,j\in I} H_1(H_2(v_i)\|j)^{x_{i,j}} \cdot u^{\mu}, v)$$
$$= e(\sigma, g) = e(\sigma', g) = e(\prod_{i,j\in I} H_1(H_2(v_i)\|j)^{x_{i,j}} \cdot u^{\mu'}, v)$$

We have:

$$1 = \prod_{i,j\in I} u^{\Delta\mu} = g^{-\beta\sum_{i,j\in I}\Delta\mu_{i,j}} \,.\, h^{\gamma\sum_{i,j\in I}\Delta\mu_{i,j}}$$

The solution of this discrete logarithm problem is as below:

$$h = g^{-\frac{\beta\sum_{i,j\in I}\Delta\mu i,j}{\gamma\sum_{i,j\in I}\Delta\mu i,j}}$$

Same as Game 2, the denominator is zero with a probability of $1/p$, which is negligible. If there is only negligible difference of the probabilities between Game 3 and Game 2, we can construct a simulator that uses the adversary to compute the discrete logarithms.

Thus, if the signature scheme is secure, two hash functions are collision-resistant. Computational Diffie-Hellman and discrete logarithm are hard in bilinear groups. There is only negligible difference in the success rate between these games.

This completes the proof.

C. SECURITY OF VERIFIABLE DYNAMIC GRAPH OPERATIONS

Theorem 9: If the Computational Diffie-Hellman is hard in bilinear group, the two hash functions (H_1, H_2) *are collision-resistant and the signature scheme is existentially unforgeable. Then in random oracle no p.p.t adversarial can convince the client to pass the verification for any fault dynamic graph operation with non-negligible probability.*

Proof: According to Theorem 6 we know that the tuple $\{ID(v_i)||n(v_i)||H_1(v_i)), \Omega_i\}$ returned from the server can pass the authentication only if it is computed on the exact v_i client queried. Based on that we can start an analysis of verifying a dynamic graph operation $OP = \{Type, v'_i, sigauth\}$ as follows:

1. If Type is U or D (in our case deletion operation is equivalent to update operation), then after update the computed with $\{v''_i\}$ different from $\{v'_i\}$ and Ω_i then R'' will be different from R' which is computed with $\{v'_i\}$ and Ω_i. In this case the verification will fail, except for $\mathcal{A}$ find collision for H_2;
2. If type is I, as we discussed before there are three type of insertion oeprations:1) Type 1 insertion, in this case the AAI Ω_i remains the same thus same as case 1 if there are any fault in the changed content, the verification will fail;2) Type 2 insertion, here the traversal labels in the posterities need to be updated, these operations can be partitioned into the update operation of a series

of v_i thus if there are any fault in the changed content, the verification will fail; 3) Type 3 insertion operation is same as update operation. In conclusion, client can detect any fault occurs in dynamic operations through VB-MHT root verification which completes the proof.

VII. DISCUSSIONS AND EXTENSIONS

A. DESIGN FOR DATA PRIVACY

To achieve data privacy and confidentiality, a straightforward way is to encrypt the data with symmetric key before sending it to the cloud. But in some applications especially cloud computing, encryption may not be necessary, and it is also resource-consuming. Therefore, some existing solutions introduce zero-knowledge proof to avoid information leakage during the interactions between TPA and server. Our scheme can be easily enhanced to preserve the data privacy. According to [6], in the *Challenge&&Proof* process, besides the challenge *chal*, the TPA also picks up a random number $\beta \overset{s}{\leftarrow} \mathbb{Z}_p^*$ and computes $POK\{(\beta)c_1 = g^\beta \wedge c_2 = e(u, v)^\beta\}$. Then it sends both POK and *chal* to the server. Later on when responding to the challenge prover, it sends $\pi = H_2(e(\sigma, c_1)c_2^{-\mu})$ back to the TPA, which can verify it based on the following principle:

$$\pi \overset{?}{=} H_2(\prod_{i,j\in I} e(H_1(H_2(v_i)||j \mod n(v_i))^{x_{i,j}}, v^\beta))$$

By using this protocol, the TPA can not obtain σ and μ from the prover. Thus it prevents possible information leakage. However, in our scheme, since we need to support dynamic graph operations, we use the structure $H_2(v_i)||j \mod n(v_i)$ which leaks the hash value of a vertex and the number of blocks in a vertex to the TPA. Such a piece of information may help a malicious TPA to more easily perform the brute-force attacks. For example, if $n(v_i)$ is small, it is easy for the TPA to retrieve the original data block. In this case, for small-size vertexes (defined by the security parameter), a random number needs to be attached to avoid off-line guess.

B. ENHANCE OUR SCHEME FOR MORE EFFICIENT VERIFICATIONS

For the basic integrity verification process, if the above design for data privacy is adopted, the communication overhead can be highly reduced. For the verification of dynamic graph operations, each time one edge or the content in vertex v_i is changed, all the corresponding authentication tags $\sigma_{i,j}$ for all sectors $v_{i,j} \in v_i$ will also be updated. Therefore, the communication overhead is high, compared to the size of changed content. To make this process more efficient, we can slightly modify the authentication tag as $\sigma_i = (H_1(H_2(v_i)||n(v_i)) \prod_{j\in n(v_i)} u_j^{v_{i,j}})^\alpha$, here $\{u_j\}$ is a set of random numbers from $\mathbb{G}_1$. In this way there will be only one tag for each v_i, and when some sectors $v_{i,n}$ in v_i are modified as $\Delta v_{i,n}$, the client only needs to compute and send $\sigma'_i = \sigma_i \cdot (H_1(H_2(v'_i)||n(v'_i)/H_1(H_2(v_i)||n(v_i))u_j^{\Delta v_{i,n}/v_{i,n}}$. In this way, only one tag needs to be sent, which reduces the communication complexity.

VIII. PERFORMANCE ANALYSIS

A. NUMERICAL ANALYSIS

Here we provide the numerical analysis of computation and communication cost from the viewpoints of the client, TPA and cloud, respectively.

1) COMPUTATION COST

Assume there is a graph $G(V, E)$, V is the number of vertexes in graph and E is the number of edges in G. After a graph G is encoded, it can be considered as being stored with n blocks.

a: DEFAULT VERIFICATION

For the client, the computation cost includes four parts: *KeyGen*, *GVD_StrGain*, *Sigag* and *TagGen*. *KeyGen* has the computation cost of (1). The MBFS complexity is linear to the total number of vertexes and edges with the cost of *GVD_StrGain* is $(V + E)$. The *Sigag* cost is (V) for MHT construction. For *TagGen*, it generates the tags for each block, and thus the cost of *TagGen* is (n). Since $n > V + E$, in total the computation cost for the client will be (n). But it is one-time cost. The TPA handles challenge generation and proof verifications. For each challenge, we assume that the detection rate for detecting r corrupted blocks is set to p. Then the TPA needs to generate c challenges such that $p = 1 - (1 - r)^c$. Here if p and r are predefined, then the complexity will be (1). Regarding proof verification, there are two parts of verifications for a MHT: computing proofs - $(\log(V))$, and checking proof - (1). Thus the computation cost for TPA is $(\log(V))$. To respond to a challenge, the cloud needs to compute the aggregated signature (σ, μ) and the hash-AAI tuple, which requires $(\log(V))$ complexity.

b: DYNAMIC GRAPH OPERATION

For update and deletion of a vertex, the main computation overhead at the client side is from the computation of the root of MHT with AAI. Such a cost is $(\log(V/vb_{size}) + \log(vb_{size}))$ here vb_{size} is the average size of Voronoi blocks in the graph; and for an insertion operation, the computation overhead in the best case will be $(\log(V/vb_{size}))$ for inserting a new voro; in the worst case, it will be the repetition of the traversal on all blocks in one Voronoi block and the computation cost will be (vb_{size}).

2) COMMUNICATION COST

a: DEFAULT VERIFICATION

During *Challenge&&Response*, in the challenge phase the TPA sends *chal* messages. The number of challenges is fixed and relatively small compared to the size of whole graph. Thus the communication overhead is (1). The response from the cloud server includes the hash-AAI tuple and has the overhead of $(\log(V/vb_{size}) + \log(vb_{size}))$.

b: DYNAMIC GRAPH OPERATION

For one update or deletion the main communication overhead is the hash-AAI tuple, which is $(\log(V/vb_{size}) + \log(vb_{size}))$.

TABLE 1. Results for real world graph datasets.

Graph	V	E	Cost
email-EuAll	265,214	420,045	3.23
soc-Epinions1	420,045	508,837	4.65
Slashdot0902	82,168	948,464	5.30
Amazon0302	262,111	1,234,877	7.54
Amazon0312	400,727	3,200,440	18.05
Wiki-Talk	2,394,385	5,021,410	36.35

For insertion, the best case is inserting a new Voronoi block and the communication cost for verifying is $(\log(V/vb_{size}))$, and the worst case is AAI for the whole block and original data in a Voronoi block. Since the block size is fixed, the communication overhead is $(\log(V/vb_{size}) + \log(vb_{size}))$.

B. SIMULATION RESULTS

Our implementation is conducted by using Java on a single core of a 4.0Ghz Intel i7-6700K with 16 GB of RAM. The field operations are implemented based on Java-Pairing-based Cryptography (jpbc) [30] library. In our implementation we use the type A curve from the library with 160-bit order $\mathbb{Z}_p$ and 512-bit order $\mathbb{G}$. All results are conducted through the average of 10 trails. Our simulation focus on the additional computation cost for supporting structure integrity and the communication cost during verifying dynamic graph operations of our scheme compare to a representative public auditing scheme [12].

In the first part, we test the computation cost for the clients to setup the system, which includes the following 5 steps: *Setup*, *KeyGen*, *GVD_StrGain*, *Sigag*, *TagGen*. Since the computation cost for *Setup*, *KeyGen*, *TagGen* is similar to other existing BLS-Based RDIC system [12], here we only test the extra cost from structure integrity and soundness assurance indices. It involves the following two steps: *GVD_StrGain*, *Sigag*. Here we set the block size limitation parameters b_{max} to $50,000$, d_{max} to 30. In Table 1 we list some real-world graphs [29] from the Stanford Large Network Dataset Collection, together with their sizes and time cost for constructing GVD-MHT. To better describe the result, we also use some random generated graphs by using the tools from [31]. In Fig. 7 we can see that the computation cost increases almost linearly as the size of graph (the number of vertexes and edges) grows. For a graph with $11,000,000$ edges and vertexes, the cost will be $52.5s$ more than the scheme in [12]. Since *GVD_StrGain* and root digest computation are not encryption-related, they can be done off-line. If the computation in cloud is considered as trustworthy by employing the verifiable outsourcing computation [32]–[34], these computations can be delegated to the cloud which can make the cost almost the same as other BLS-based RDIC schemes. Also, in later simulation results we will show how this construction help reduce the communication cost for verifying dynamic graph operations.

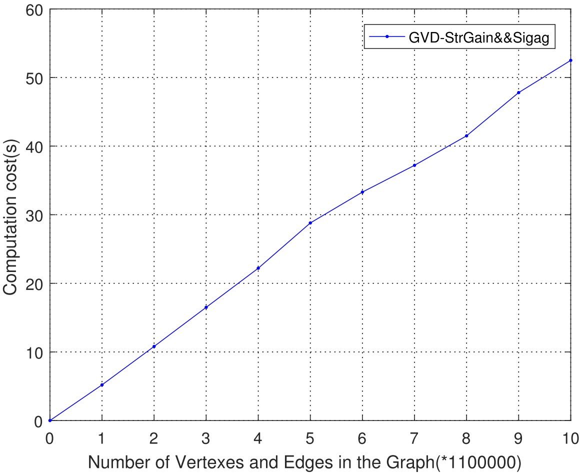

FIGURE 7. Additional computation overhead for supporting structural integrity verification with different sizes of graphs.

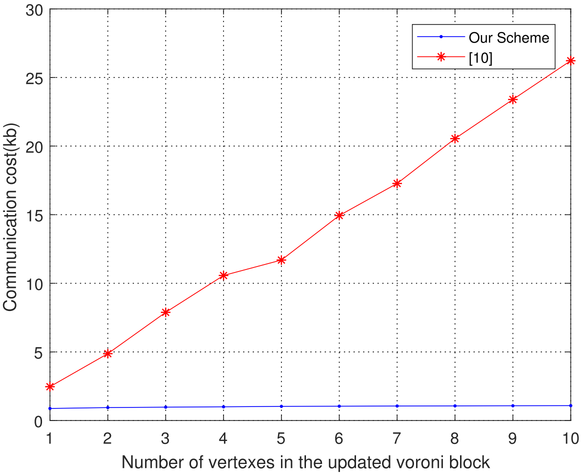

FIGURE 8. Size of response from server for one verifiable dynamic operation on one Voronoi block.

In the second experiment, we investigated the communication cost for verifiable dynamic graph operations, the test is established on a graph with 11,000,000 vertexes and edges. Since the updated data and authentication tag is same for both our scheme and [12], here we focus on the size of data that need to be retrieved from server. Results are shown in Fig. 8 and Fig. 9. In real world graph applications, one update always involves a connected sub graph, this type of update can be considered as a combination of updating several Voronoi blocks and vertexes. As shown in Fig. 8, the size of data that need to be retrieved from server for updating, deleting or inserting a Voronoi block is significantly reduced. It is clear that our scheme scales well while the number of vertexes in the updated Voronoi blocks increases. From Fig. 9, we can see that the server response for updating or inserting a vertex is also reduced by adopting our scheme. Since in our scheme all sectors belong to one vertexes are in the same subtree in the bottom level, as the number of sectors in one vertexes increases the length of server response increase slightly.

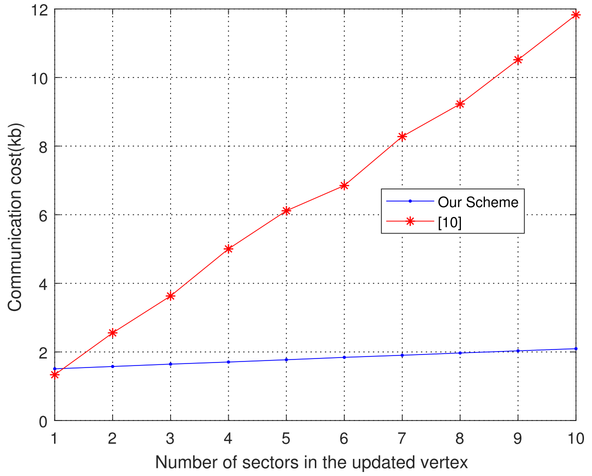

FIGURE 9. Size of response from server for one verifiable dynamic operation on one vertex.

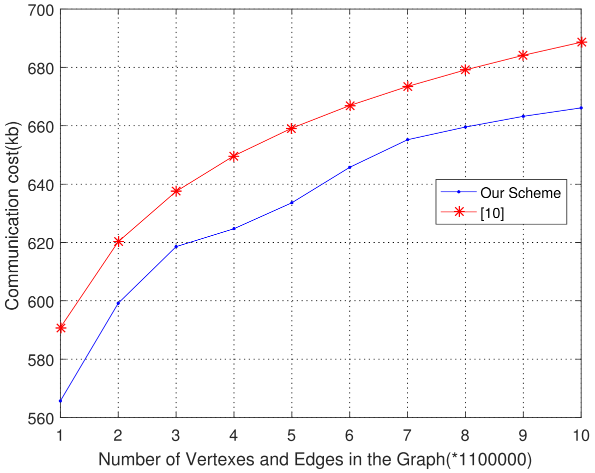

FIGURE 10. Communication cost for default auditing process.

Our results indicate that the communication overheads for verifying subgraph update in our scheme has significant advantage compared to the scheme in [12].

We also test the communication cost during default auditing process here we set the number of sectors queried each time as 460 which can provide 99 percent detetection . The result is shown in Figure 10. As we can see, the cost also grows slowly as the size of graph increases, and the growth is also due to the change of the MHT size.

Based on these simulation results, we can conclude that our scheme has significant advantage in auditing remote big graph. The size of data required to be retrieved from cloud for verifying subgraph dynamic operations is only slightly affected by the size of subgraph and vertexes in the subgraph. The trade of for the advantage on verifying subgraph dynamic operations and support for structural integrity is the increased computation cost during the construction process which can be done offline or delegate to cloud.

IX. CONCLUSIONS AND FUTURE WORKS

In this work, we have proposed a new RDIC scheme, i.e., GVD-based remote graph integrity checking for securing the big graph in cloud storage. This scheme also supports verifiable dynamic graph operations including update, insertion and deletion. Our construction process utilized the GVD to isolate dynamic operations inside each Voronoi block to achieve efficient verification. Moreover, we have introduced an additional step into the default auditing process to ensure that the server does not cheat with an intact block for querying against a corrupted block. We have provided theoretical proofs to show that it achieves soundness and completeness. Both numerical analysis and simulation results showed that our scheme is cost-efficient and scalable for real-world big graphs.

If our scheme is used in multi-user environment and the graphs are divided into subgraphs with different access rules, the data privacy may not hold during the process of verifying dynamic graph operations. One of the open issues for our future work is to support fine-grained access control of subgraphs with efficient user revocation and join. In addition, our scheme adopts BLS signature to reduce the signature size. However, BLS is based on bilinear map, and its computation cost is higher than the signatures based on ECC. Therefore, the implementation based on ECC and the study of how its performance differs from BLS-based scheme is another open problem for our future works.

REFERENCES

[1] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 135–146.

[2] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein. (2014). "GraphLab: A new framework for parallel machine learning." [Online]. Available: https://arxiv.org/abs/1408.2041

[3] C. Avery, "Giraph: Large-scale graph processing infrastructure on Hadoop," in *Proc. 26th Conf. Uncertainty Artif. Intell. (UAI)*, Catalina Island, CA, USA, Jul. 2010.

[4] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[5] D. Hubbard *et al.*, "Top threats to cloud computing v1.0," Cloud Secur. Alliance, Seattle, WA, USA, White Paper, 2010, pp. 1–14. [Online]. Available: www.cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

[6] Y. Yu *et al.*, "Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 4, pp. 767–778, Apr. 2017.

[7] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 598–609.

[8] G. Ateniese *et al.*, "Remote data checking using provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, 2011, Art. no. 12.

[9] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Springer, 2008, pp. 90–107.

[10] S. Qiu, G. Xu, H. Ahmad, and L. Wang, "A robust mutual authentication scheme based on elliptic curve cryptography for telecare medical information systems," *IEEE Access*, vol. 6, pp. 7452–7463, 2018.

[11] S. Qiu, G. Xu, H. Ahmad, and Y. Guo, "An enhanced password authentication scheme for session initiation protocol with perfect forward secrecy," *PLoS ONE*, vol. 13, no. 3, p. e0194072, 2018.

[12] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 847–859, May 2011.

[13] Y. Yu *et al.*, "Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage," *Int. J. Inf. Secur.*, vol. 14, no. 4, pp. 307–318, 2015.

[14] A. Kundu and E. Bertino, "Privacy-preserving authentication of trees and graphs," *Int. J. Inf. Secur.*, vol. 12, no. 6, pp. 467–494, 2013.

[15] R. C. Merkle, "A certified digital signature," in *Proc. Conf. Theory Appl. Cryptol.* Springer, 1989, pp. 218–238.

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

[17] M. U. Arshad, A. Kundu, E. Bertino, K. Madhavan, and A. Ghafoor, "Security of graph data: Hashing schemes and definitions," in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy*, 2014, pp. 223–234.

[18] D. Yan, J. Cheng, Y. Lu, and W. Ng, "Blogel: A block-centric framework for distributed computation on real-world graphs," *Proc. VLDB Endowment*, vol. 7, no. 14, pp. 1981–1992, 2014.

[19] Y. Lu, F. Hu, and X. Li, "Towards the security of big data: Building a scalable hash scheme for big graph," in *Proc. 14th Int. Conf. Inf. Technol., New Gener.* Las Vegas, NV, USA: Springer, 2017, pp. 235–243.

[20] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[21] C. Liu *et al.*, "Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2234–2244, Sep. 2014.

[22] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. 4th Int. Conf. Secur. Privacy Commun. Netw.*, 2008, Art. no. 9.

[23] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 17, no. 4, 2015, Art. no. 15.

[24] S. K. Goel, C. Clifton, and A. Rosenthal, "Derived access control specification for XML," in *Proc. ACM Workshop XML Secur.*, 2003, pp. 1–14.

[25] A. F. Barsoum and M. A. Hasan, "Provable multicopy dynamic data possession in cloud computing systems," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 3, pp. 485–497, Mar. 2015.

[26] C. Liu, R. Ranjan, C. Yang, X. Zhang, L. Wang, and J. Chen, "MuR-DPA: Top-down levelled multi-replica Merkle hash tree based secure public auditing for dynamic big data storage on cloud," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2609–2622, Sep. 2015.

[27] W. Liu, X. Liu, J. Liu, Q. Wu, J. Zhang, and Y. Li, "Auditing and revocation enabled role-based access control over outsourced private EHRs," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun. (HPCC), IEEE 7th Int. Symp. Cyberspace Saf. Secur. (CSS), IEEE 12th Int. Conf. Embedded Softw. Syst. (ICESS)*, Aug. 2015, pp. 336–341.

[28] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 131–143, Jan. 2013.

[29] Q. Liao, "On Reed-Solomon codes," *Chin. Ann. Math. B*, vol. 32, pp. 89–98, 2010.

[30] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun./Jul. 2011, pp. 850–855.

[31] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Math.*, vol. 6, no. 1, pp. 29–123, 2009.

[32] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno, "Hash first, argue later: Adaptive verifiable computations on outsourced data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1304–1316.

[33] C. Costello *et al.*, "Geppetto: Versatile verifiable computation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2015, pp. 253–270.

[34] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2013, pp. 238–252.

YU LU received the B.E. degree in electronic engineering from Chongqing University, Chongqing, China, in 2014. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL, USA. His current research interests include wireless mesh networks and big data security.

FEI HU received the Ph.D. degree in signal processing from Tongji University, Shanghai, China, in 1999, and the Ph.D. degree in electrical and computer engineering from Clarkson University, New York, NY, USA, in 2002. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, The University of Alabama, Tuscaloosa, AL, USA. He has published over 150 journal/conference papers and book (chapters). His research has been supported by the U.S. NSF, Cisco, Sprint, and other sources. His current research interests include cyber-physical system security and medical security issues, intelligent signal processing, such as using machine learning algorithms to process sensing signals, and issues on wireless sensor network design.