A Survey on Data Integrity Checking and Enhancing Security for Cloud to Fog Computing

K.Uma Maheswari
Department of CSE
National Institute of Technology
Tiruchirappalli 620015, India
406118003@nitt.edu

S.Mary Saira Bhanu
Department of CSE
National Institute of Technology
Tiruchirappalli 620015, India
msb@nitt.edu

S.Nickolas
Department of CA
National Institute of Technology
Tiruchirappalli 620015, India
nickolas@nitt.edu

***Abstract*—Cloud computing plays an important role in the computing world as it provides a cost-effective way of delivering services to customers through the internet. The Cloud Service Providers (CSPs) offer an enormous amount of storage space with low cost and eliminates the data management burden of the customers. As the operational details are not transparent, the CSPs may be untrusted. They may steal the user data, tamper the data, remove the rarely accessed data or hide the data loss by duplicating the data. In such cases, data integrity fails and needs to be addressed. The emerging fog computing paradigm extends the cloud platform to provide services for IoT applications. So the data integrity verification techniques are not only to be incorporated over the cloud but also to be extended for fog computing. This survey discusses the various data integrity checking methodologies for both cloud and fog environment.**

***Index Terms*—data integrity, cloud computing, fog computing, authentication structure**

I. Introduction

Most of the organizations stored their data in data centers but building and maintaining such infrastructures are not an economical solution. So, the organizations outsourced their data storage in centralized data centers. Due to the emergence of cloud technology, organizations are moving towards the cloud for their data storage. Cloud storage service is a boon to organizations as it provides unlimited storage for low cost, scalable and location independent platforms for managing customer data. It stores data on the remote servers which are maintained by CSPs and follows the pay-as-you-go model.

Cloud storage reduces the capital cost, application deployment time, information management and maintenance but it raises security challenges like confidentiality, availability and integrity [1] since the data are not under the control of the user. Confidentiality means protecting the information from being accessed by unauthorized users. It can be attained by encrypting and storing the data in a cloud server and decrypting it whenever needed. Availability means the ability of users to access information without any interruption. It can be attained by replicating the data in redundant servers. The major issue in cloud environment is maintaining data integrity which ensures the consistency, accuracy and completeness of data. When the customers start outsourcing their data to the cloud, there are chances of data corruption; data might get deleted either due to technical glitch or due to administration errors. Hence, the users must ensure that the data stored in the servers are correct [2]. Researchers work to overcome this issue by enabling the user to check the completeness of data with data integrity verification methodologies.

Traditional techniques (RSA, MD5) that are used for integrity verification require the user to download the data, compute the metadata and verify it with the already stored one. This creates an unnecessary burden for the user with limited capabilities [3]. To solve this, a challenge-response protocol is designed. In this approach, the user computes and stores metadata locally then challenges the server to compute the metadata and sends it back for verification. As it adds additional burden to the user, the verification task is handed over to the Third-Party Auditor (TPA) which is known as public auditing. In this, the user data are divided into multiple blocks and for each block, the metadata is computed. An authentication structure is constructed using the computed metadata and it is maintained by TPA for integrity verification.

With the emergence of the Internet of Things (IoT), the usage of IoT devices also increases in cloud environment. IoT devices have very less memory and low processing capabilities, they need some nearby data centers to store and process their data. The time-sensitive IoT applications require low-latency network connections between endpoints i.e. between IoT devices and servers where the data are processed. In such cases, centralized cloud data centers are not suitable. This issue is addressed by fog computing which brings the potential of the cloud closer to the place where the data are generated [4]. This changes the data-centric cloud environment into a distributed and localized environment.

Fog computing was initiated by Cisco in January 2014 to ease the wireless data transfer to the distributed devices in the IoT network paradigm. In fog computing, data are processed near the place from where it is generated, so that the amount of bandwidth needed to transfer the data from the IoT devices to the cloud is reduced drastically. Huge volumes of data are filtered, aggregated, processed, analyzed and the necessary data alone are sent to the cloud [5]. Therefore, network traffic is reduced to a great extent. In time-sensitive applications, fog nodes process large volumes of data and trigger the IoT devices according to the processed data to produce an instant response to the user.

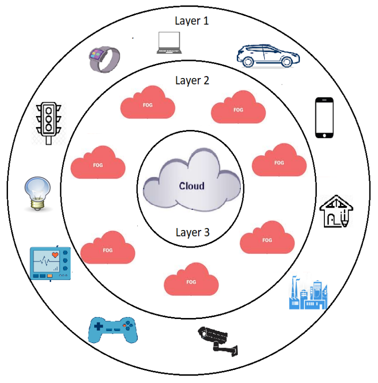

Fig. 1. Three layered fog computing architecture.

Fig. 1 depicts the three-layered architecture of fog computing. Layer 1 is the end devices layer that consists of IoT enabled devices such as sensors, smartphones, smartwatches, tablets, etc. Layer 2 is the fog layer that consists of routers, gateways, switches, access points, proxy servers, base stations, etc., termed as fog nodes, have limited computation and storage facilities. Layer 3 is the cloud layer that consists of cloud servers, cloud data centers which have sufficient computation and storage resources [6]. There exist three service models in fog computing: (a) offloading model: IoT data are offloaded to the nearest fog node, (b) aggregation model: data streams from IoT devices are aggregated by fog nodes, (c) peer-to-peer model: the nearest fog nodes share their resources [7].

In the fog environment, both fog and cloud server maintains the database but compared to cloud server the fog server stores minimal data that are frequently used. Whenever the end user requests data, either the fog server or cloud server responds to it with the available data and the end user should receive the data without any distortions [8]. For example, in a health care application, if a patient detail is requested, the corresponding fog server responds to it. This response should reach the doctor without any diversification. Otherwise, the doctor suggests the wrong prescription to the patient [9]. Thus, integrity preservation plays a major role in data storage and retrieval in a fog environment.

When the data are stored on fog nodes, the users lose their control on those data similar to the cloud. As a result, some malicious users may modify or delete the data to destroy some evidence [4]. Also, fog nodes may discard the rarely accessed data to reclaim the storage space. So, it is unavoidable to ensure the integrity of data stored on fog nodes. But, the problem with fog platform is fog nodes are not capable of performing complex computational operations and also IoT devices generate large volumes of data with high velocity [10] and to manipulate this enormous amount of data yields heavy network traffic, needs improved algorithms to maintain data integrity. This paper discusses the authentication structures used in the existing integrity verification protocols and their limitations.

The rest of the paper is organized as follows: Section II describes the system model of the data integrity verification. Section III classifies the integrity checking mechanisms based on the authentication structure used. Section IV concludes the survey.

II. System Model

Cloud storage allows the user to store their data on remote servers and access it quickly with minimal cost. The software industries move their data from the local environment to the cloud data centers to reduce the storage, maintenance and management cost. But the outsourcing of data into the cloud makes the user data vulnerable to several threats such as data alteration, data duplication and data deletion. So, it is mandatory to verify the integrity of user data stored in cloud storage to ensure its correctness.

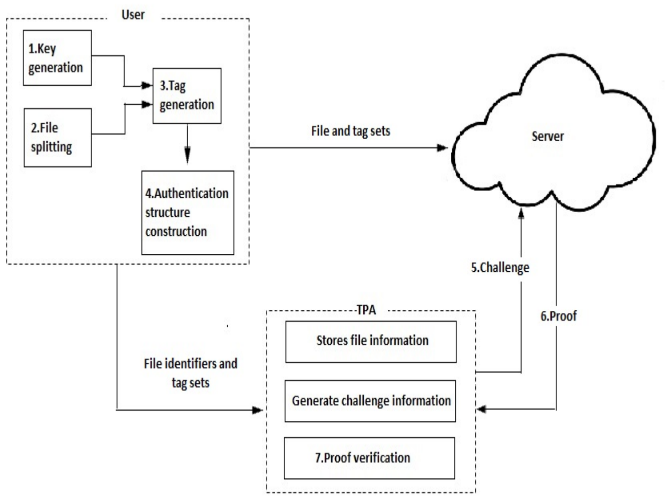

Fig. 2. System structure.

Fig. 2 depicts the standard methodology to verify the integrity of cloud storage. The following are the steps used in cloud data integrity verification: 1. Key generation : In this phase, techniques like bilinear mapping are used to generate key pairs: Public key and Secret key. These keys are not based on a single parameter; they are the combination of multiple parameters. 2. File splitting : The user file is divided into blocks and in some cases, the blocks are again divided into sectors. 3. Tag generation : The tag is generated for each block or each sector using public, private key and some random information. 4. Authentication structure construction : The tag values of each block are used for the construction of the authentication structure. The file, tag values and partial key information are sent to the server. The authentication structure or the signed value extracted from the structure is stored in the TPA or user side. 5. TPA/User challenges the server : The verifier challenges the server by sending random block indices. 6. Proof generation : The server generates proof information as a response and sends it to the verifier. 7. Proof verifiability : The TPA verifies the proof by using the stored file information.

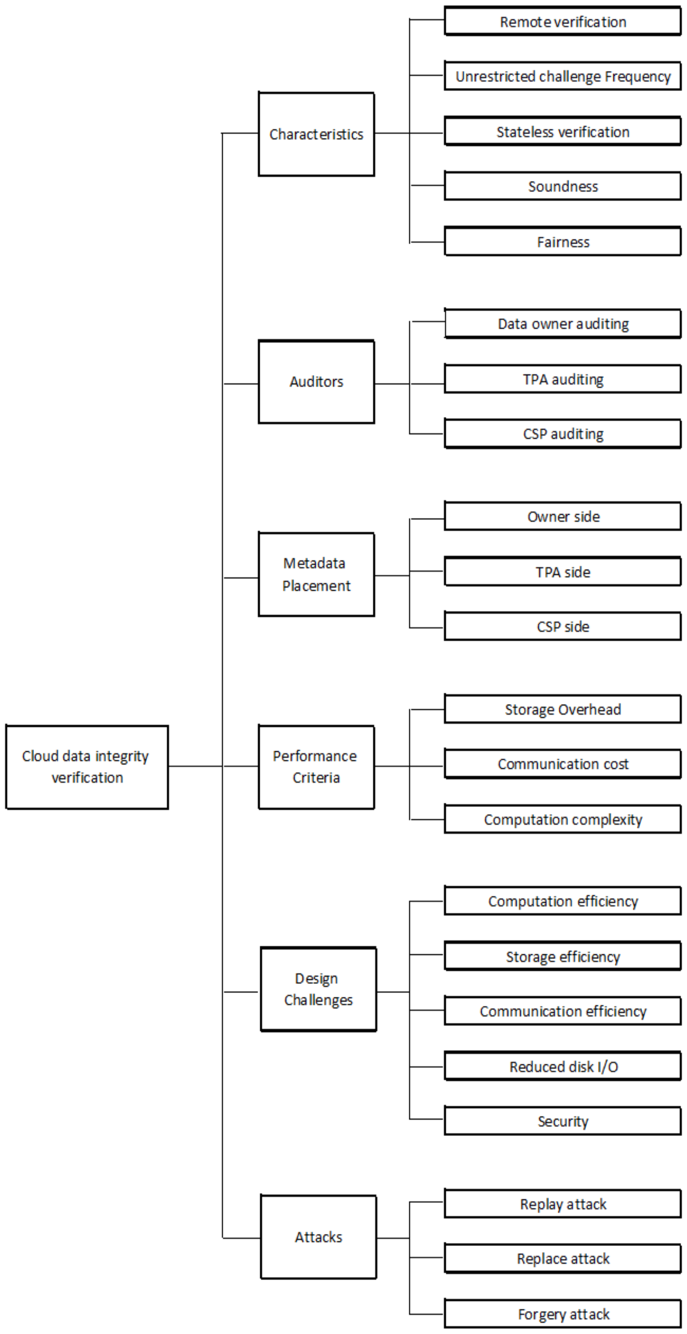

Fig. 3. Aspects of cloud data integrity verification methodology.

Fig. 3 shows several aspects of data integrity verification methodology. It outlines the characteristics of the verification protocol, auditors involved in the verification, metadata placement, performance criteria, design challenges and possible attacks in verification. Usually, the verification protocol should verify the integrity of the user data without bringing it to the user/TPA side and there should not be any restrictions on the number of challenges. It should protect the honest CSP and should detect even smaller corruption also. The auditor may be a Data Owner (DO), a TPA, or a CSP. Since the main purpose of the cloud is to reduce the maintenance overhead of the user, it is meaningless to hand over the verification task to the DO. On the other side, if CSP is involved in the verification, one cannot expect an honest result. So, the verification task is left with the TPA. If the metadata calculated from the file is stored on the user side, the communication cost will be high. To reduce the cost, it is stored in the TPA side. The verification protocol should be storage efficient, communication efficient, cost-efficient and the number of block access should be a minimum one [11].

III. Classification of Cloud Data Integrity Verification Schemes

Data integrity can be verified by different techniques. Most of them divide the file into multiple blocks, generate the secret information for each block and store it in the auditor side for verification. Many researchers are working on cloud data integrity verification schemes. Zhou et al., [3] classify the integrity verification techniques according to the number of users and the number of copies maintained in the cloud storage. Dong et al., [12] classify the integrity checking protocols into ID-based encryption and non-ID based encryption. Yang et al., [13] classify the auditing methods as MAC-based, RSA-based and BLS-based. Zafar et al., [11] summarize the characteristics, design challenges and issues of data integrity verification schemes.

This survey classifies the integrity verification mechanisms based on the authentication structures used and the classifications are root signature-based, Bloom Filter (BF) based, table-based, tag regeneration-based. Classification through authentication structure is worthwhile in finding a suitable structure for maintaining integrity in single and multiple copies of static and dynamic data. The fundamental techniques behind these schemes are cryptographic functions such as encryption, decryption, hash value calculation and signature computation.

A. Root Signature-based Scheme

This scheme is based on the tree structure in which the tree is constructed from leaves to root. MHT (Merkle Hash Tree) is a basic tree structure used in most of the verification schemes [3]. Hashed Multiple Branches Tree (HMBT), Multiple Hash Tree (MPHT), MHT ternary tree [14], [15], [16] are the variations of MHT. There are two primary phases in the verification scheme: setup phase and verification phase.

In the setup phase, the user file is divided into blocks and for each block, a signature is computed using hash functions. Then a tree-based structure is constructed, where the leaf nodes are used to hold the hash values of the file blocks in a left-to-right sequence and the non-leaf nodes are used to hold the combined hash values of its children. The user calculates the root value of the tree structure and signs the root using the private key. The signed root along with the file, file identifier, tag sets of file blocks and some secret information are sent to the CSP. The CSP receives the information and computes the root by constructing a tree using that information. The newly calculated root is compared with the root sent by the user. If they match, then the CSP stores all the information sent by the user in the cloud server. Otherwise, the CSP asks the user to resend the information. The signed root, file identifier and some secret information are stored in the TPA side for integrity verification.

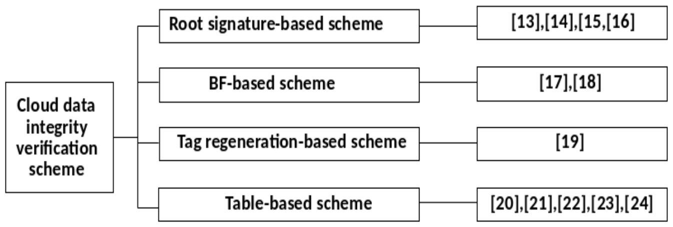
Cloud data integrity verification scheme
Root signature-based scheme
[13],[14],[15,[16]
BF-based scheme
[17],[18]
Tag regeneration-based scheme
[19]
Table-based scheme
[20],[21],[22],[23],[24]

Fig. 4. Taxonomy of cloud data integrity verification schemes.

In the verification phase, the TPA generates challenge information by choosing a random set of blocks and random numbers. This challenge information is sent to the server. The server replies with the signed root, hash values of the blocks to be checked and auxiliary information i.e. hash values of the siblings. The TPA constructs the tree and computes the root value with the received information and checks it with the already stored one. If the verification fails, then the TPA informs to the user otherwise the TPA repeats the same verification procedure.

Wang et al., [14] have presented a scheme for ensuring the integrity of data stored in the cloud storage based on MHT, bilinear mapping and BLS signature. The user generates the MHT using the block signatures, computes the root, signs the root and sends the signed root, block signatures, file to the server. During auditing, the user picks a random set of blocks and sends the block numbers to the server. The server replies with the signed root, hash values of the blocks to be verified and hash values of the siblings. It ensures remote data integrity, provides public auditability and dynamic operations support.

Shao et al., [15] have achieved storage security in cloud computing using HMBT structure, BLS signature and bilinear mapping. HMBT differs from MHT in terms of the number of child nodes. MHT follows a binary tree where HMBT follows an n-ary tree. Hash values of each basic block are used to construct a tree and the signed root is used for integrity verification. Since the height of the MBHT structure is shorter than other tree structures, it reduces the calculation burden of the user and improves the efficiency of constructing and querying the authentication structure.

Another form of MHT known as MPHT is implemented by Fu et al., [16] for integrity verification. In s-fork MPHT structure, every node has s child nodes except the leaf nodes and stores five values: index of the node, height of the node, number of leaf nodes that can be traversed, children of the node and value of the node. The coded data blocks which are in matrix form are generated by multiplying the data blocks with a coefficient matrix. Each row of a coded data matrix is stored in a separate cloud server with its signed MPHT root. This scheme verifies the integrity of stored data and retrieves the corrupted data from healthy servers. It requires fewer resources for verification and the integrity tags are unforgeable.

In paper [17], Zhao et al., enforced data integrity using Rank Based Skip List (RBSL). The rank of the ith node in the RBSL is the number of bottom nodes that can be reachable from the current node. The root of the RBSL is computed using the hash values of each block and the public key. The receipt signed by both the parties (user and CSP) after storing the file protects them from denying each other.

Since this scheme is based on the tree structure, dynamic operations such as block insertion, deletion, modification can be easily done within a file and they effectively support integrity verification. But it requires more cryptographic hash operations to generate the tree when there are multiple copies of a file exist.

B. BF-based Scheme

BF is a data structure that is based on the theory of probability and is constructed using hash functions. In the setup phase, the user file is divided into blocks and the BF is constructed by applying the keyed hash functions to each block. BFs generated from all the data blocks are consolidated as a single BF and kept it for verification. In the verification phase, an auditor (TPA/DO) randomly picks some block numbers and sends it to the cloud server for verification. The server first applies the keyed hash function to each block to get the BFs for challenged blocks. The server combines all the BFs and sends it to the verifier. The verifier processes it using the private key and compares it with the stored one. If there is a match, then the auditor concludes that the data integrity is preserved.

In the case of cloud databases BF is constructed for each tuple or each column of a database and it is stored in an encrypted form. During query retrieval, the user can verify the integrity of the database by reconstructing the BF and comparing it with the stored one. It creates a burden to the user when BF is constructed for each tuple. Likewise, it is difficult for the user to verify the integrity when BF is created for each column of a database.

Jeong et al., [18] have aimed to impose a data consistency in the IoT environment using BF. In this scheme, the codeword is generated for each file block using a private key and the hash set. This codeword is used to set the bits of a BF. Either the TPA or the DO stores the BF for verification. This scheme is suitable for processing large amounts of data and it saves processing time since it does not include any key generation mechanism. The data alteration detection probability is low because of false positives.

In paper [19], Ferreti et al., used BF for data integrity verification in cloud databases. Multiple keyed hash functions are applied on the tuples. The resulting hash values are used to set the bits of the BF and the BF string is encrypted using a secure symmetric encryption algorithm. The encrypted BF strings are stored in the last column of each tuple and integrity verification is part of the query retrieval. Whenever the client issues a select or update query, the values are retrieved along with the digest. First, the user decrypts the digest and then verifies the integrity by applying the membership test. This scheme supports null value insertion in a database. It involves lesser computation and minimal network and storage overhead. But it cannot prevent adaptive attacks.

C. Tag Regeneration-based Scheme

This scheme ensures the data integrity in the fog-to-cloud based IoT scenario. In this, tag generation is a two-step process. In the first step, the data collected from the IoT devices are organized into data blocks and tags are generated by mobile sinks. In fog environment, mobile sinks are data sources for fog nodes and they have sufficient storage capacity. They collect data from various sensors, group into blocks, generate tags and forward them to the fog node. In the second step, fog nodes verify the tags generated by mobile sinks and generate a new set of tags for each block that can be used for integrity verification.

Tag regeneration is first introduced by Tian et al. [20] for integrity verification in IoT-fog-cloud environment. In this scheme, each mobile sink selects a random number as a private key and computes its public key. Likewise, each fog node selects a private key and generates a public key. The Mobile sink generates tags for all the blocks and uploads it to the fog node.

Fog node verifies the integrity of each block and generates new sets of tags using data blocks and tags generated by mobile sinks. Fog node sends data blocks and newly generated tags to the CSP. The CSP verifies its integrity and stores it. During auditing, TPA generates a challenge message and sends it to the CSP. The response proof is generated by the CSP and sent to the TPA for verification. This scheme reduces the communication cost, computational cost, and energy consumption in the verification phase. Identity privacy is achieved through fog nodes.

D. Table-based Scheme

This scheme is based on a table structure located at TPA to record the file information for dynamic auditing. The table structure records the changes of data blocks and achieves rapid auditing and efficient data updating. This structure is helpful to trace the recent Version Information (VI) of the file. Doubly Linked Info Table (DLIT), Dynamic Hash Table (DHT), Index Logic Table (ILT), Map-Version Table (MVT) and Index Hash Table (IHT) [21], [22], [23], [24], [25] are some of the table structures used in integrity verification.

In the setup phase, the user splits the file into blocks, constructs the table structure that stores the block number, version number of each block and some other secret information. The block number uniquely identifies the block and the version number indicates whether the block is updated or not. Initially, the version number is set to 1 and it is incremented when the block is updated. A hash value is calculated for each block using a table structure and a verifiable tag is calculated using this hash value. The file blocks and tag sets are sent to CSP, table structure and some other secret information are sent to TPA.

During the verification phase, TPA constructs a challenge message by choosing n block indices randomly along with n random coefficients and sends it to the CSP. The CSP calculates the response and sends to the TPA. TPA verifies it using the table structure. During dynamic operations such as insertion, deletion, updation, the user modifies the file, makes the changes in the table structure according to modifications, finds new hash values, calculates new tags to modified blocks and sends newly calculated tags, modified blocks to CSP. This verification procedure is repeated periodically.

Shen et al., [22] addressed the integrity verification using DLIT and Location Array (LA). DLIT is a two-dimensional data structure stored by the TPA to verify the cloud data integrity. DLIT consists of two parts: File information and Block information. File information consists of file ID and user ID to identify each file. Block information includes the version number and the timestamp of each block. Location array acts as a mapper between the index number and the physical location of data blocks. The file ID, user ID, version information, timestamp and location information are uploaded to TPA. Using this information, the TPA constructs DLIT and LA which are used for verification.

Tian et al., [23] have proposed a cloud data integrity scheme based on DHT. The file information is stored in DHT and there are two elements in DHT: File element and Block element. The file element is implemented as an array-like structure. It stores the file identifier and the pointer to a first block element. The block element is implemented as a linked list. Each node of a block element stores the current version of the block, timestamp and a pointer to a next node. This table is stored in the TPA side. The modification of a file needs only to update the linked list associated with that file. This scheme provides blockless verification by Homomorphic Verifiable Authenticators (HVAs), privacy protection by random masking and batch auditing by BLS signature scheme. The problem with this scheme is it takes more time to search operation.

Barsoum et al., [24] have proposed a scheme to check the integrity of multiple copies of a file using a single MVT. Since it is needed to construct a tree for every copy of a file, integrity verification of multiple copies of a file using a tree-based structure is not an efficient one. In this case, MVT is a cost-effective data structure that can be implemented using a linked list and consists of two fields: block number and version number. In this scheme, encrypted file blocks are divided into sectors and a random secret is selected for each block to avert tag forgery attack. For each block, tag is computed using the hash value generated from each entry of the MVT, random secret and file block. An aggregated tag for blocks at the same indices in each copy is calculated. File identifier, aggregated tag sets, file copies are sent to CSP and MVT is sent to TPA for integrity verification.

In [21], Wang et al., designed a scheme for dynamic provable data possession based on ILT. This scheme uses ILT with MHT. The index number of the file block is stored in the first field and the logical number of the file block is stored in the second field of the ILT. The values in each row are used for the calculation of block tags. This scheme has lower computation cost in the insertion, deletion and updation process. This scheme is proposed to overcome the problem associated with the MVT.

Zhu et al., [25] have presented a scheme that uses IHT as an authentication structure for dynamic data integrity verification. For each block, tags are calculated using hash values of that block and the block-tag pairs are stored in the CSP side where the encrypted secrets are stored as Publicly Verifiable Parameters (PVP) in the TPA side. With this PVP, TPA stores file information like block number, version number, and random integer in IHT for integrity verification. IHT structure can reduce the extra storage needed for tags by increasing the number of sectors in each block. Since the IHT follows the sequence structure, the insertion and deletion operations need to change the sequence numbers of blocks and recalculation of tags which in turn induces more computational cost and communication overhead. Tables I, II, III and IV show the methodologies and limitations of the integrity verification schemes.

TABLE I
ROOT SIGNATURE-BASED SCHEME

S.No	Title	Methodology	Limitations
1	DIPOR: An IDA-based Dynamic Proof of Retrievability Scheme for Cloud Storage Systems, 2018	MPHT, Coefficient matrix, Information dispersal algorithm and Bilinear mapping	Supports modification operation only. Insertion and deletion of a data block is not possible.
2	Dynamic Data Integrity Auditing Method Supporting Privacy Protection in Vehicular Cloud Environment, 2018	HMBT, BLS signature and Bilinear mapping.	Compared to MHT, the depth of the Multiple branches tree is minimal, but each node is having multiple siblings. So communication cost is high.
3	Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing, 2010	MHT, BLS signature and Bilinear mapping.	Incurs high computational and communication overhead
4	User Stateless Privacy-preserving TPA Auditing Scheme for Cloud Storage, 2019	RBSL and Bilinear mapping	Construction of skip list needs extra pairings and hash functions.

IV. CONCLUSION

The intent of this survey is to study the various techniques based on the authentication structures used for data integrity verification in the cloud and fog environment and to analyze their pitfalls. This paper gives the classification of various integrity verification schemes. The classifications are root signature-based, BF-based, tag regeneration-based and table-based schemes. This survey concludes that root signature-based schemes are suitable for integrity verification in a single

TABLE II
BF-BASED SCHEME

S.No	Title	Methodology	Limitations
1	Secure Cloud Storage Service Using Bloom Filters for the Internet of Things, 2019	BF with multiple public keys and private keys.	Data alteration detection probability is low because of false positives.
2	A Symmetric Cryptographic Scheme for Data Integrity Verification in Cloud Databases, 2018	BF and Secure symmetric encryption algorithm.	Cannot prevent adaptive attacks. Protects only the digest but not the data.

TABLE III
TAG REGENERATION-BASED SCHEME

S.No	Title	Methodology	Limitations
1	Privacy-preserving Public Auditing for Secure Data Storage in Fog-to-cloud Computing, 2019	Tags regenerated by fog nodes and Bilinear mapping.	Tag regeneration involves computation and storage overhead

TABLE IV
TABLE-BASED SCHEME

S.No	Title	Methodology	Limitations
1	Identity-based Non-repudiable Dynamic Provable Data Possession in Cloud Storage, 2018	ILT and Bilinear mapping.	Checks only the integrity. No discussion about retrievability of corrupted data.
2	An Efficient Public Auditing Protocol with Novel Dynamic Structure for Cloud Data, 2017	DLIT, Location array, BLS signature and Bilinear mapping.	Expensive cryptographic methods make the system to work very slowly.
3	Dynamic Hash Table Based Public Auditing for Secure Cloud Storage, 2015	DHT, Homomorphic authenticator and Bilinear mapping.	Takes more time to search operation.
4	Provable Multicopy Dynamic Data Possession in Cloud Computing Systems, 2014	MVT and Bilinear mapping.	Suffered by delete-insert attack.
5	Dynamic Audit Services for Outsourced Storage in Clouds, 2011	IHT, Random sampling and Bilinear mapping.	Insertion and deletion operations need to change the sequence numbers of blocks and recalculation of tags. It induces more computational cost and communication overhead.

copy of dynamic data, BF-based schemes are suitable for a single copy of static data, tag regeneration-based schemes are suitable for IoT data and table-based schemes are suitable for multiple copies of dynamic data. Existing cloud data integrity verification methods are not suitable for fog environment because of its cryptographic nature. So there is a need for a lightweight scheme to verify the data integrity in the fog environment. In the future, a lightweight protocol based on erasure coding techniques will be proposed for integrity verification and corrupted data retrieval in the fog environment.

References

[1] L. A. Mohammed and K. Munir, "Secure third party auditor (tpa) for ensuring data integrity in fog computing," *International Journal of Network Security & Its Applications (IJNSA) Vol*, vol. 10, 2018.

[2] S. Chakraborty, S. Singh, and S. Thokchom, "Integrity checking using third party auditor in cloud storage," in *2018 Eleventh International Conference on Contemporary Computing (IC3)*. IEEE, 2018, pp. 1–6.

[3] L. Zhou, A. Fu, S. Yu, M. Su, and B. Kuang, "Data integrity verification of the outsourced big data in the cloud environment: A survey," *Journal of Network and Computer Applications*, vol. 122, pp. 1–15, 2018.

[4] J. Ni, K. Zhang, X. Lin, and X. S. Shen, "Securing fog computing for internet of things applications: Challenges and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 601–628, 2017.

[5] S. Khan, S. Parkinson, and Y. Qin, "Fog computing security: a review of current applications and security solutions," *Journal of Cloud Computing*, vol. 6, no. 1, p. 19, 2017.

[6] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury, and V. Kumar, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, pp. 19 293–19 304, 2017.

[7] V. Moysiadis, P. Sarigiannidis, and I. Moscholios, "Towards distributed data management in fog computing," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[8] S. Kunal, A. Saha, and R. Amin, "An overview of cloud-fog computing: Architectures, applications with security challenges," *Security and Privacy*, vol. 2, no. 4, p. e72, 2019.

[9] A. Alazeb and B. Panda, "Ensuring data integrity in fog computing based health-care systems," in *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 2019, pp. 63–77.

[10] S. Sagiroglu and D. Sinanc, "Big data: A review," in *2013 international conference on collaboration technologies and systems (CTS)*. IEEE, 2013, pp. 42–47.

[11] F. Zafar, A. Khan, S. U. R. Malik, M. Ahmed, A. Anjum, M. I. Khan, N. Javed, M. Alam, and F. Jamil, "A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends," *Computers & Security*, vol. 65, pp. 29–49, 2017.

[12] Y. Dong, L. Sun, D. Liu, M. Feng, and T. Miao, "A survey on data integrity checking in cloud," in *2018 1st International Cognitive Cities Conference (IC3)*. IEEE, 2018, pp. 109–113.

[13] K. Yang and X. Jia, "Data storage auditing service in cloud computing: challenges, methods and opportunities," *World Wide Web*, vol. 15, no. 4, pp. 409–428, 2012.

[14] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE transactions on parallel and distributed systems*, vol. 22, no. 5, pp. 847–859, 2010.

[15] B. Shao, G. Bian, Y. Wang, S. Su, and C. Guo, "Dynamic data integrity auditing method supporting privacy protection in vehicular cloud environment," *IEEE Access*, vol. 6, pp. 43 785–43 797, 2018.

[16] A. Fu, Y. Li, S. Yu, Y. Yu, and G. Zhang, "Dipor: An ida-based dynamic proof of retrievability scheme for cloud storage systems," *Journal of Network and Computer Applications*, vol. 104, pp. 97–106, 2018.

[17] H. Zhao, X. Yao, X. Zheng, T. Qiu, and H. Ning, "User stateless privacy-preserving tpa auditing scheme for cloud storage," *Journal of Network and Computer Applications*, vol. 129, pp. 62–70, 2019.

[18] J. Jeong, J. W. J. Joo, Y. Lee, and Y. Son, "Secure cloud storage service using bloom filters for the internet of things," *IEEE Access*, vol. 7, pp. 60 897–60 907, 2019.

[19] L. Ferretti, M. Marchetti, M. Andreolini, and M. Colajanni, "A symmetric cryptographic scheme for data integrity verification in cloud databases," *Information Sciences*, vol. 422, pp. 497–515, 2018.

[20] H. Tian, F. Nan, C.-C. Chang, Y. Huang, J. Lu, and Y. Du, "Privacy-preserving public auditing for secure data storage in fog-to-cloud computing," *Journal of Network and Computer Applications*, vol. 127, pp. 59–69, 2019.

[21] F. Wang, L. Xu, H. Wang, and Z. Chen, "Identity-based non-repudiable dynamic provable data possession in cloud storage," *Computers & Electrical Engineering*, vol. 69, pp. 521–533, 2018.

[22] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2402–2415, 2017.

[23] H. Tian, Y. Chen, C.-C. Chang, H. Jiang, Y. Huang, Y. Chen, and J. Liu, "Dynamic-hash-table based public auditing for secure cloud storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 701–714, 2015.

[24] A. F. Barsoum and M. A. Hasan, "Provable multicopy dynamic data possession in cloud computing systems," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 3, pp. 485–497, 2014.

[25] Y. Zhu, G.-J. Ahn, H. Hu, S. S. Yau, H. G. An, and C.-J. Hu, "Dynamic audit services for outsourced storages in clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 2, pp. 227–238, 2011.