MuR-DPA: Top-down Levelled Multi-replica Merkle Hash Tree Based Secure Public Auditing for Dynamic Big Data Storage on Cloud

Chang Liu, Rajiv Ranjan, Chi Yang, Xuyun Zhang, Lizhe Wang, Jinjun Chen

Abstract—Big data and its applications are attracting more and more research interests in recent years. As the new generation distributed computing platform, cloud computing is believed to be the most potent platform. With the data no longer under users' direct control, data security in cloud computing is becoming one of the most obstacles of the proliferation of cloud. In order to improve service reliability and availability, storing multiple replicas along with original datasets is a common strategy for cloud service providers. Public data auditing schemes allow users to verify their outsourced data storage without having to retrieve the whole dataset. However, existing data auditing techniques suffers from efficiency and security problems. First, for dynamic datasets with multiple replicas, the communication overhead for update verification is very large, because verification for each update requires O(log*n*) communication complexity and update of all replicas. Second, to the best of our knowledge, there is no existing integrity verification schemes can provide public auditing and authentication of block indices at the same time. Without authentication of block indices, the server can build a valid proof based on data blocks other than the block client requested to verify. In order to address these problems, in this paper, we present a novel public auditing scheme named MuR-DPA. The new scheme incorporated a novel authenticated data structure based on the Merkle hash tree, which we name as MR-MHT. For support of full dynamic data updates, authentication of block indices and efficient verification of updates for multiple replicas at the same time, the level values of nodes in MR-MHT are generated in a top-down order, and all replica blocks for each data block are organized into a same replica sub-tree. Compared to existing integrity verification and public auditing schemes, theoretical analysis and experimental results show that the MuR-DPA scheme can not only incur much less communication overhead for both update and verification of datasets with multiple replicas, but also provide enhanced security against dishonest cloud service providers.

Index Terms—Big Data, Cloud Computing, Data Security, Public Auditing, Replica Management

1 Introduction

BIG data has been one of the most intensive research topics in recent years. People from almost all major industries are increasingly realizing the values in their explosively growing datasets. Research directions for big data are always into 4 v's: Velocity, Variety, Veracity and Volume, in which cloud can help in a big way. Cloud computing is the new-generation distributed computing platform that is extremely useful for big data storage and processing. With the pay-as-you-go payment model, elastic and scalable resource allocation and various levels of services in IaaS (Infrastructure-as-a-Service), PaaS and SaaS, cloud is widely recognised as the most promising technological backbone for solving big data problems [5]. Cloud can also save a lot of investments in purchasing and maintenance of hardware, which is also great for big data applications. A vision is that cloud, providing computational resources, can one day be integrated into our daily life so close as other utilities such as electricity, gas and water [11].

Security/privacy is one of the major concerns in the usage of cloud computing [19, 26]. As data are no longer under users' direct control, users are reluctant to move their valuable data onto cloud, especially public cloud with high consolidation and multi-tenancy. Also, from an efficiency perspective, querying and retrieving with cloud server requires a lot more efforts than with local servers.

Datasets in big data applications are always dynamic. In fact, except for a few examples of large static datasets such as libraries and e-archives, datasets in most big data applications needs constant updating. In many applications data updates are very frequent, such as in social networks and business transactions. Therefore, it is of extreme importance for a cloud security mechanism, such as a public auditing scheme, to efficiently support dynamic data.

Three main dimensions in security are confidentiality, integrity and availability. Aiming at integrity assurance, public auditing of cloud data has been an extensively investigated research problem in recent years. As user datasets stored on cloud storage servers (CSS) are out of the user's reach, auditing from the client herself or a third party auditor is a common request, no matter how powerful the server-side mechanisms claim to be. With provable data possession (PDP) and proofs of retrieveability (POR), the data owner or a third-party auditor can verify integrity of their data without having to retrieve their data. In such schemes, a small metadata called 'homomorphic authenticator' or 'homomorphic tags' are stored along with each data block. When the client needs to verify data integrity, the server will generate a proof with the authenticators of the selected data blocks, and data auditing is done by the client or a third-party auditor through verifying the proof with the public keys.

Existing public auditing schemes can already support

public auditing and various kinds of full dynamic data updates at the same time [17, 31]. However, there are a few problems that we aim to address in this work. First, not much work has been done in support multiple replicas. Storing multiple replicas is a common strategy for reliability and availability in cloud. For highly dynamic data, each update will lead to update to every replica. Given the fact that update verifications in current verification schemes are of O(log$n$) communication complexity, verifying these replicas one by one will be very costly in terms of communication. Second, current schemes for dynamic public auditing are susceptible to attacks from dishonest servers. Although there is an integrity verification schemes for dataset with replicas [13] and schemes with index verification [14], they cannot support public auditing.

In this paper, we present a multi-replica dynamic public auditing (MuR-DPA) scheme that can bridge the gaps mentioned above through a newly designed authenticated data structure. Research contributions of this paper can be summarized as follows:

1. To address the efficiency problem in verifiable updates for cloud storage with multiple replicas, we propose a multi-replica public auditing (MuR-DPA) scheme. The new scheme is based on a novel multi-replica Merkle hash tree (MR-MHT). Experimental results show that our scheme can drastically reduce communication overheads for update verification of cloud data storage with multiple replicas.

2. As the previous usage of Merkle hash tree (MHT) in public auditing of dynamic data did not involve authentication of node indices, such schemes are susceptible to cheating behaviours from a dishonest server. In this paper, with the support of MR-MHT, we proposed the first MHT-based public auditing scheme for dynamic data with authentication of index information, which is safe against dishonest servers. The main strategy is top-down levelling and verification of indices from both sides.

3. With MR-MHT, we also designed a novel public auditing protocol for verification of all replicas at once. Experimental results show that our scheme can not only provide efficient verification for multiple replicas, but also incur less extra storage overhead at server side.

Paper Organization: The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 provides an analysis of our research problem. Section 4 provides a detailed description of our proposed scheme in detail. Section 5 provides security and efficiency analysis for our design. Section 6 provides experimental results. Section 7 provides conclusion for this research.

2 Related Work

Compared to traditional systems, scalability and elasticity are key advantages of cloud [1, 5, 11]. As such, efficiency in supporting dynamic data is of great importance. Security and privacy protection on dynamic data has been studied extensively in the past [9, 14, 15, 31]. Frequent updates exist in many cloud applications such as business transaction logs, health records from hospitals and online social networks (e.g. Twitter [21]).

Data security/privacy is one of the most pressing concerns related to big data and cloud [22, 33, 37]. There is a lot of research to enhance cloud data security/privacy with technological approaches on cloud server side, such as [18, 34]. They are of equal importance as external verification approaches such as our focus of public auditing.

Integrity verification for outsourced data storage has attracted extensive research interest. The concept of proofs of retrievability (POR) and its first model was proposed by Jules, et al. [16]. Unfortunately, their scheme can only be applied to static data storage such as archive or library. In the same year, Ateniese, et al. proposed a similar model named 'provable data possession' (PDP) [7]. Their schemes offer 'blockless verification' which means the verifier can verify the integrity of a proportion of the outsourced file through verifying a combination of precomputed file tags which they call homomorphic verifiable tags (HVTs) or homomorphic linear authenticators (HLAs). Work by Shacham, et al. [23] provided an improved POR model with stateless verification. They also proposed the first public verification scheme in the literature that based on BLS signature scheme [10]. In this scheme, the generation and verification of integrity proofs are similar to signing and verification of BLS signatures. When wielding the same security strength (say, 80-bit security), a BLS signature (160 bit) is much shorter than an RSA signature (1024 bit), which in turn brings shorter proofs for a POR scheme. They also proved the security of both their schemes and the PDP scheme by Ateniese, et al. [6, 7]. Ateniese, et al. extended their scheme for enhanced scalability [9], but only partial data dynamics and a predefined number of challenges is supported.

Erway, et al. proposed the first PDP scheme that can support verification for full dynamic data updates [14]. A modified authenticated data structure (ADS) is used for verification of updates, which became the popular way of supporting verifiable updates in the following PDP/POR works. The ADS they used is called rank-based authenticated skip list (RASL). However, public auditability and variable-sized file blocks are not supported in their framework. Wang, et al. [31] proposed a scheme based on BLS signature that can support public auditing (especially from a third-party auditor, TPA) and full data dynamics. To support verification of updates, they used another ADS called Merkle hash tree (MHT). However, their usage of ADS was flawed, which will be patched in this work. A follow-up work by Wang et al. [30] added a random masking technology on top of [31] to ensure the TPA cannot infer the raw data file from a series of integrity proofs. In their scheme, they also incorporated a strategy first proposed in [23] to segment file blocks into multiple 'sectors'. for trading-off of storage and communication costs. Work by Liu et, al. [17] investigated support for fine-grained updates and efficiency for verification of small updates. However, their scheme is under a strong assumption, where they assumed the server remains hon-

est answering queries to file blocks. Also, none of the above schemes has considered the commonly employed multi-replica strategy in clouds.

For cloud storage with multiple replicas, Curtmola, et al. [13] proposed a scheme named MR-PDP that can prove the integrity of multiple replicas along with the original data file. Although the scheme only requires only one authenticator for each block, it has two severe drawbacks. First, it does not support public auditing, which means the verifications can only be done by the client herself. Second, it does not support dynamic data. To-date, to allow a third-party auditor to verify datasets with multiple replicas, the client needs to store and build different ADS for every replica.

Research in this area also includes the work of Ateniese, et al. [8] on how to transform a mutual identification protocol to a PDP scheme; scheme by Zhu, et al. [36] which allows different service providers in a hybrid cloud to cooperatively prove data integrity to data owner. As cloud data sharing is happening in many scenarios, Wang et, al. worked on secure data verification of shared data storage [27] and also with efficient user management [29] and user privacy protection [28]. Zhang et, al. proposed a scheme with a new data structure called update tree [35]. Without conventional authenticated data structures such as MHT, the proposed scheme has a constant proof size and support fully data dynamics. However, the scheme also does not support public auditing. Cash et, al. [12] proposed a novel POR scheme based on oblivious RAM (ORAM). ORAM, or oblivious file system, was mostly used to hide data access patterns through shuffling and noise addition on outsourced data storage [25, 32]. Shi et, al. also proposed a more efficient scheme based on ORAM [24], but such schemes are still not practical to use.

3 Problem Statement and Analysis

3.1 Multiple Replicas

For availability, storing multiple replicas is a default setting for cloud service providers. Storing replicas at different servers and/or locations will make user data easily recoverable from service failures. A straightforward way for users to verify the integrity of multiple replicas is to store them as separate files and verify them one by one. Currently, the most common technique used to support dynamic data is authenticated data structure (ADS). Given the $\log(n)$ communication complexity and storage complexity of ADS (n is the total number of blocks, a very large number when file is large), different replicas. More importantly, an update for each data block will require update of the corresponding block in every replica. If all replicas are indexed in their own separated ADS, the client must verify these updates one by one to maintain verifiability. The 'proof of update' for each block contains $\log(n)$ hash values as auxiliary authentication information (AAI). Therefore, the communication cost in update verifications will easily become a disaster for users whose cloud datasets are highly dynamic. In previous schemes, researchers have considered support for public auditing, data dynamics and multiple replicas, but none has considered them all together.In this work, we try to address this problem with a new ADS which links together all replicas for each block.

In [13], the authors proposed a multi-replica verification scheme with great efficiency by associating only one authenticator (HLA) for each block and all replica blocks. Although this approach can bring great benefits such as lower storage cost at server side and less pre-processing time at client side, their scheme do not support public auditability. The verification needs the privately kept padding randoms (or at least the pseudo-random function that used to generate them). If leaked, another party will know how to compute the original message based on it and how to compute an arbitrary replica based on an original file block, which is also the inherent reason why this setting cannot even be transferred into a scheme with public verifiability.

To sum up, from our considerations, desired properties of a multi-replica verification scheme should include the following:

1. Public Auditability and Support for Dynamic Data -- Enables a third-party auditor to do the regular verification for the client and allow the client to verify data updates. It will be unreasonable for the client to conduct verification herself on a regular basis, where she only wants to know when something went wrong about her data; and dynamic data exists in most applications.

2. All-round Auditing -- Enables efficient verification for all replicas at once so that the verifier will get better confidence. If any of the replicas fails, the server will be notified on time.

3. Single-Replica Auditing -- Enables verification for an arbitrary replica for some specific blocks; because the verifier may only wants to know if at least one replica is intact for less important data.

3.2 Secure Dynamic Public Auditing

As demonstrated in Fig.1, the three parties in a public auditing game -- the client, the cloud service provider and third-party auditor -- are not fully trusted. by each other, Authenticated data structures (ADS) such as MHT or RASL can enable other parties to verify the content and updates of data blocks. The authentication for a block is accomplished with the data node itself and its auxiliary authentication information (AAI) which is constructed with node values on or near its verification path. Without verification of block indices, a dishonest server can easily take another intact block and its AAI to fake a proof that could pass authentication. This will cause several security holes. First, the proofs of updates are no longer reliable. A dishonest server can store new data block anywhere, as long as it transfers back a consistent pair of hash $H(m_i)$ and AAI that can be used to compute the correct root value. Second, for auditing of dynamic data, $H(m_i)$, the hash value of the block itself, is needed in authenticator computation instead of hash of any value that contains block indices such as $H(i)$ or $H(v||i)$, otherwise an in-

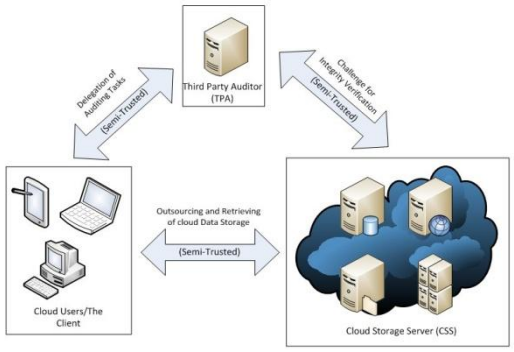

Fig. 1. Relations between the participating parties in public auditing of cloud data

sert/delete will cause change of authenticators of all following blocks, which will be disastrous, especially that the client is the only one who can compute authenticators. Therefore, in order for each authenticator to include a block-specific hash value, $H(m_i)$ seemed to be the only choice. In this case, as the verifier (client or TPA) does not possess the original dataset, the client will solely rely on cloud server -- who keeps the actual dataset -- to compute $H(m_i)$ for verification of data integrity. As the only way for the client to verify the correctness of $H(m_i)$ is through ADS, the server can cheat the client with another hash and AAI pair. In other words, the server can take any other healthy block to replace the block that should be verified, which denies the primary aim of integrity verification. To the best of our knowledge, there is no existing public auditing scheme that supports full dynamic data can deal with this problem.

Erway et, al.'s RASL [14] can provide authentication for indices, which is resilient to the above attacks. Aside from the effective ADS, they did propose a scheme where the authenticator is computed as $T = g^m$, but it is too simple (without hash value, they can be integrated/separated too easy) to support public auditing. In fact, the RASL cannot be directly applied into a public auditing scheme supporting dynamic data. As stated earlier, $H(m_i)$ is to be used in authenticators for support of dynamic data. Therefore, the client needs $H(m_i)$ computed by and transferred from the cloud server for verification. In order to achieve verifiability of index information, the leaf nodes no longer stores the hash value of file blocks, but the hash value of a concatenation of multiple values in the form of $f(v) = H(l(v), r(v), x(v), f(\text{rgt}(v)))$. Therefore, the server need to send back both values of $f(v)$ and $H(m)$, and the client will need to verify $H(m)$. In an RASL, a common case is that multiple leaf nodes are in the same verification path, such as n_3, n_4, n_5 in Fig. 2. Let's say n_3, n_4, n_5 represents message blocks m_3, m_4, m_5. As stated earlier, the client needs $H(m_i)$ computed by and transferred from the cloud server for verification. In this case, if verification of m_3 is needed, the server not only needs to return all 3 values on n_3, n_4, n_5 as part of AAI, but also needs to compute and transfer all $H(m_3), H(m_4)$ and $H(m_5)$. As only a small fraction of blocks (460 for 99% confidence when auditing 1GB file), it

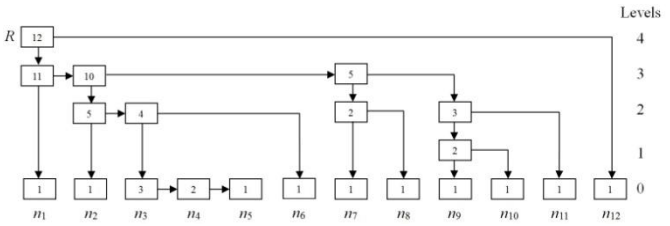

Fig. 2. A Rank-based Authenticated Skip List (RASL)

is not likely that these consecutive blocks are chosen for one audit, which means much excessive overheads. Also, the bottom-up levelling restricts the insertions. If leaf nodes are level 0 as defined in [14], any insertion that creates a new level below level 0 will cause update of all level values (therefore all hash values of all nodes), which is hardly possible for the client to verify. For these reasons, we choose to use MHT with top-down levelling, instead of RASL, to design the new ADS. Now that the leaf nodes are on different levels, we will need both the client and verifier to remember the total number of blocks and verify the block index from both directions (leftmost to rightmost, rightmost to leftmost) to make sure the server do not cheat the client with another node on the verification path.

4 MuR-DPA

4.1 Preliminaries

4.1.1 Bilinear Pairing

Assume a group G is a gap Diffie-Hellman (GDH) group with prime order p. A bilinear map is a map constructed as $e: G \times G \to G_T$ where G_T is a multiplicative cyclic group with prime order. A useful e should have the following properties:

1. Bilinearity – $\forall\, m, n \in G \Rightarrow e(m^a, n^b) = e(m, n)^{ab}$;
2. Non-degeneracy – $\forall m \in G, m \neq 0 \Rightarrow e(m, m) \neq 1$; and
3. Computability – e should be efficiently computable.

As denoted in [10], a more efficient asymmetric bilinear map $e: G_1 \times G_2 \to G_T$ may also be applied. For simplicity, we will use this symmetric bilinear map in our scheme description.

4.1.2 Merkle Hash Tree

The Merkle Hash Tree (MHT) [20] has been intensively studied in the past. Similar to a binary tree, each node N will have a maximum of 2 child nodes. In fact, according to the update algorithm, every non-leaf node will constantly have two child nodes. Information contained in one node N in an MHT T is constructed as follows. For a leaf node based on a file block m_i, node value is computed as $h_i = H(m_i)$. A parent node of N_1 and N_2 is constructed as $N_P = \{H(h_1||h_2)\}$. A leaf node m_i's auxiliary authentication information (AAI) Ω_i is a set of hash values chosen from every of its upper level so that the root value R can be computed through $\{m_i, \Omega_i\}$.

4.2 MuR-DPA: Multi-replica Dynamic Public

Auditing

4.2.1 MR-MHT

A multi-replica Merkle hash tree (MR-MHT) is a novel authenticated data structure designed for efficient verification of data updates, as well as authentication for block indices. Each MR-MHT is constructed based on not only a logically segmented file, but also all its replicas, as well as a pre-defined cryptographic hash function H. An example of MR-MHT, constructed based on a file with 4 blocks and 3 replicas, is shown in Fig. 3. The differences from the MHT are as follows:

1. Value stored in the leaf nodes are hash values of stored replica blocks. In MR-MHT, leaf nodes represents replica blocks $b_{i,j}$, namely the jth replica of the ith file block.

2. Value stored in a node v from a none-leaf level is computed from the hash values of its child nodes and two other indices $l(v)$ and $r(v)$. $l(v)$ is the level of node v and $r(v)$ is the maximum number of nodes in the leaf(bottom) level that can be reached from v. Different to RASL in [14], the levels are defined in an top-down order, i.e., the level of root node R is defined as 0, and levels of its child nodes are defined as 1, etc.. The values stored in leaf nodes are $H(1||l(b_{i,j})||H(b_{i,j})$; the value in each none-leaf node is computed as $H(r||l||h_{left}||h_{right})$ where h_{left} and h_{right} denotes the values stored in its left child node and right child node, respectively. In Fig. 3, under our definition, $l(b_{i,j})$ (and for all leaf nodes) is 4, $r(b_{i,j}) = 1$. For example, the value h_d is computed as:

$$h_d = H\left(r(d)||l(d)||h(b_{1,1})||h(b_{1,2})\right)$$
$$= H\left(2||3||h(b_{1,1})||h(b_{1,2})\right)$$

and $h(b_{1,2}) = H\left(1||4||H(b_{1,2})\right)$, $h_b = H(6||1||h_1||h_2)$, etc..

3. The AAI $\Omega_{i,j}$ is different from the MHT in [31] as follows. They now contain not hash values of the intermediate nodes only, but tuples in the format of $\{h, l, q, d\}$, one tuple for each node. h is the hash value stored on this node, l is the level of this node, q is the maximum number of leaf nodes reachable from this node, and d is a Boolean value that indicates this node is to the right (0) or left (1) of the node on the verification path, i.e. the nodes from leaf node to the root R. For example, in Fig. 3, $\Omega_{2,1}$ for replica block $b_{2,1}$ is defined as $\{(h(b_{2,1}), 4,1,0), (h(b_{2,2}), 4,1,0), (h(b_{2,3}), 3,1,0), (h_1, 2,3,1), (h_c, 1,6,0)\}$, and its verification path is $\{h(b_{2,1}), h(e), h(2), h(b), R\}$.

4. All replicas of one file block are organized into a same sub-tree which we call replica sub-tree (RST), see Fig. 3. Note that each RST has the same structure. Each block has exactly c replicas because there are c replica files for the original data file. The total number of leaf nodes for every RST is the total replica number c. Different from [13], replica blocks are treated independently and each replica block has its own authenticator. The root of each RST, which we denote as h_i, will play a vital role in the newly proposed multi-replica verification and update verification in the following sections. We use Ω_i to denote

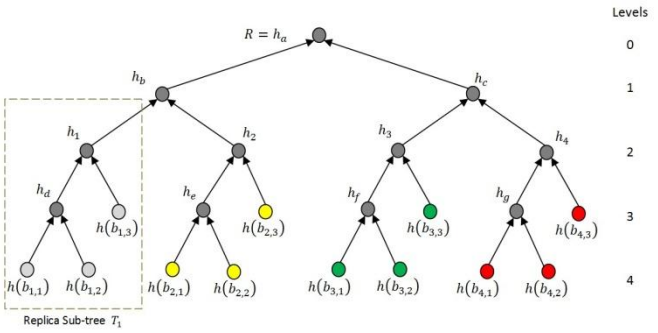

Fig. 3. An example of MR-MHT

the AAI for h_i, i.e., one can verify the content and index of h_i with Ω_i and R, similar to $\Omega_{i,j}$ discussed earlier but has less hash values. Although roots of RSTs are non-leaf nodes, they can still be authenticated in the same way as leaf nodes. In addition, we define Γ_i as the set of tuples $\{h, l, q, t\}$ for all π intermediate nodes in each RST T_i. t is the sequence number from 1 to π, ordered from top to the bottom right. For example, in Fig. 3, Γ_2 contains only one node h_e where $\Gamma_2 = \{(h_e, 3,2,1)\}$. As the number of replicas is only a small number (less than 10), for simplicity of description, we assume the structure of T_i is stored at client (and TPA) side, which applies to every RST and takes only a negligible amount of storage. In this case, the client can compute Γ_i, therefore h_i, based on $h(b_{i,j})$ and $l(b_{i,j})$ without requesting Γ_i it from the server. But keep in mind, for less client-side storage, the client may also request Γ_i from the server and verify them via $R, h(b_{i,j})$ and Ω_i .

Based on this new ADS, we now describe our scheme in detail.

4.2.2 Setup

The user and cloud server will first establish common parameters, including a bilinear map $e: G \times G \to G_T$, and a cryptographic hash function H.

$KeyGen(1^k)$: The client generates a secret value $\alpha \in \mathbb{Z}_p$ and a generator g of G, then compute $v = g^\alpha$ where v, g are the public key and α is the secret key. Another secret signing key pair $\{spk, ssk\}$ is chosen with respect to a designated provably secure signature scheme whose signing algorithm is denoted as Sig(). This algorithm outputs $\{ssk, \alpha\}$ as the secret key sk and $\{spk, v, g\}$ as the public key pk.

$FilePreProc(F, sk, c)$:

1) For a dataset to be stored on cloud server, the client will first make c replicas based on the original files. In order to enable the verifiability of these replicas, they should be different from one another; otherwise, the server may cheat the client by responding to challenges with the correct proofs but actually storing only one replica. From an original file $F = \{m_1, m_2, \dots, m_n\}$, we denote its jth replica file as $\tilde{F}_j = \{b_{1,j}, b_{2,j}, \dots, b_{n,j}\}, j \in [1, c]$. The replica blocks $b_{i,j}$ are transformed from m_i, and the transform is reversible, i.e., the client can recover the original file F through retrieval and reversed transformation of any replica $\tilde{F}_j$. Therefore, the client do not have to upload F; she can recover F with any intact replica if needed. For

example, a method described in [13] is to choose n pseudo-random functions ψ to compute random values $r_{i,j} = \psi(j||i)$ then output $b_{i,j}$ as $b_{i,j} = m_{i,j} + r_{i,j}$; the replicas may also be computed with other methods such as public-key techniques.

2) The client constructs a MR-MHT based on $b_{i,j}$, computes the root value R, and computes its signature sig with ssk.

3) The client will compute an authenticator $\sigma_{i,j} = \left(H\left(b_{i,j}\right)u^{b_{i,j}}\right)^{\alpha}$ for every replica block $b_{i,j}$.

Finally, this algorithm outputs $\{b_{i,j}, \sigma_{i,j}, sig\}$, and then uploads them all to the cloud server.

4.2.3 Data Updates and Verification

In this paper, types of updates considered are whole-block insertion I, deletion D and modification M. These are the minimum requirements for support of full data dynamics [14]. In multi-replica scenario, when a block m_i needs to be updated, all its corresponding replica blocks $b_{i,j}$ are also needed to be updated in the same way to maintain consistency. For insertion and modification, the client needs to upload new data block. As the only one that has the capability to compute replica blocks $b_{i,j}$ based on the original file block m_i, the client will compute the new replica blocks $b'_{i,j}$ then send them to the server along with the update type I, D or M.

$PerformUpdate(UpdateReq)$: The server will parse $UpdateReq$ into $\left\{Type, i, \{b'_{i,j}\}\right\}$ and perform the update to file blocks, indices and ADS according to the update request. Specifically, the server will need to update the l value for nodes in insertions and deletions. Note that values in none-leaf nodes in Ω_i stays the same after the update process.

For insertions and deletions, the situations are more complex than in past schemes [14, 17, 31]. In a traditional MHT, level or rank information is not contained in the nodes; in an RASL, all leaf nodes stays constantly on level 0. Therefore, there is no need to change the hash value in other nodes. In this top-down levelled MHT however, the levels of all leaf nodes in adjacent RST have also changed by +1 with insertion/-1 with deletion, as the level value is a part in computation of node value. For example, in Fig. 4-a, with the insertion of $\{b'_{3,j}\}$, the levels of $\{b_{3,j}\}$ have increased by 1, which will cause change to all $\{h(b_{3,j})\}$; while in Fig. 4-b, with the deletion of $\{b_{3,j}\}$, levels of the old $\{b'_{3,j}\}$ (i.e., old $\{b_{4,j}\}$) have decreased by 1. To output the correct R', these updates are needed to be performed in the hash tree as well. For insertions and modifications, The server will then output $P_{update} = \left\{\{h(b_{i,j})\}, \Omega_i, R', sig\right\}$

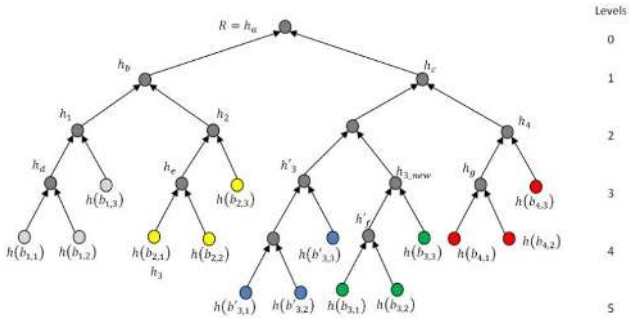

Fig. 4-a. An Insertion before the 3rd block into the MR-MHT in Fig. 3

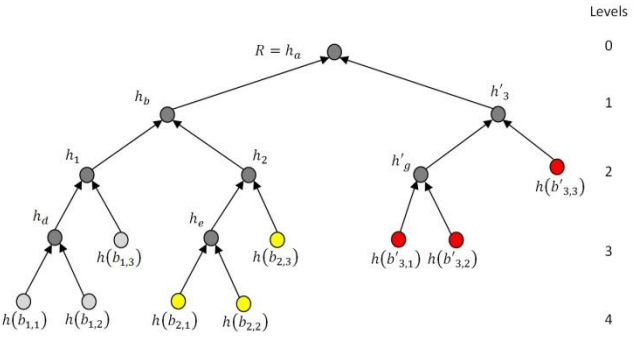

Fig. 4-b. A Deletion of the 3rd block for the MR-MHT in Fig. 3

and returns it to the client. For deletions, the server will need to additionally transfer $h'(b_{i,j})$.

$VerifyUpdate(pk, P_{update})$: In order to verify this update, the client first need to parse P_{update}. Let the π tuples in Ω_i be (f_k, l_k, q_k, d_k) for each node N_k in an decreasing order of levels, i.e., $l_1 = l(h_i), \dots, l_{\pi-1} = 2, l_\pi = 1$. A little different from the definition, q_k is the max number of RST roots, instead of leaf nodes, that can be reached from N_k. Since the structure of RST T_i is known to the client, she will be able to compute h_i and h'_i, the old and new roots of T_i, with $h(b_{i,j})$ (got from the server) and $h(b'_{i,j})$ alone respectively.

1. The client will first iteratively compute tuples $\{\lambda_k, \eta_k, \xi_k, \zeta_k\}$ for nodes on the verification path with nodes N_k in Ω_i as follows, $k = 1, \dots, \pi$:

$\lambda_k = q_{k-1} + \lambda_{k-1}$, $\eta_k = H(\lambda_k||l||f_k||\eta_{k-1})$, $\xi_k = \xi_{k-1} + q_k$ and $\zeta_k = \zeta_{k-1}$ if $d_k = 1$,

or:

$\lambda_k = q_{k-1} + \lambda_{k-1}$, $\eta_k = H(\lambda_k||l||\eta_{k-1}||f_k)$, $\xi_k = \xi_{k-1}$ and $\zeta_k = \zeta_{k-1} + q_k$, if $d_k = 0$,

where $\eta_0 = h(b_{i,j})$, $\lambda_0 = 1$, $\xi_0 = 0$, $\zeta_0 = 0$.

After$\{\lambda_\pi, \eta_\pi, \xi_\pi, \zeta_\pi\}$ is obtained, client will verify $R = \eta_\pi$ with sig, and verify if $\xi_\pi = i - 1$ and $\zeta_\pi = n - i$ hold at the same time. If the three values passed authentication, the authenticity of Ω_i (also $b_{i,j}$) and its index i can be confirmed.

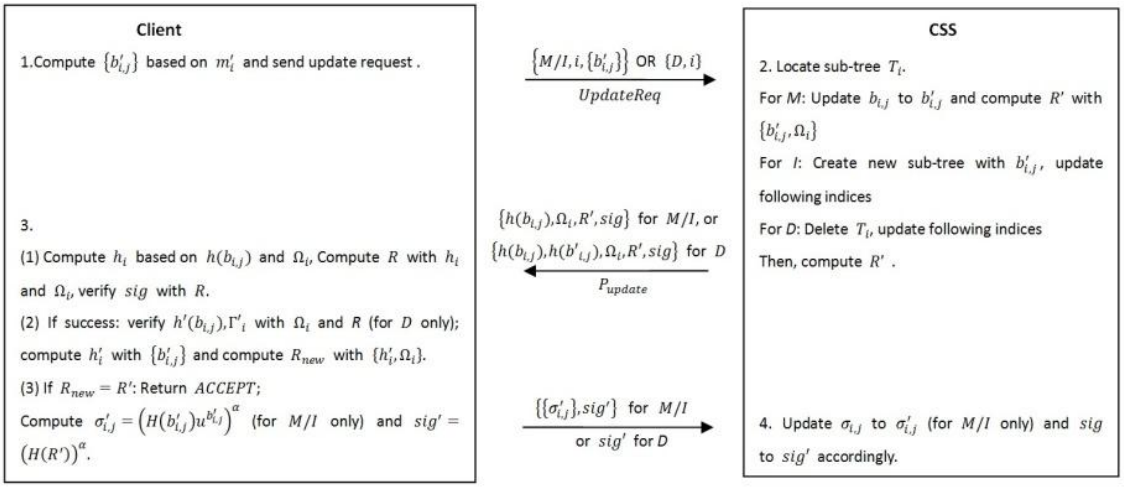

Fig. 5. Data update and verification

2. For deletion, the client needs to verify $h'(b_{i,j})$. Note that $h'(b_{i,j})$ represented the same block and replicas whose root of RST was stored as the first tuple in Ω_i, e.g., in Fig. 4-b, $h(b'_{3,j})$ and $h(b_{4,j})$ represented the same set of data; the only difference is that $l(b'_{3,j}) = l(b_{4,j}) - 1$. Therefore, the client has enough information to verify $h(b'_{i,j})$ with $h(b_{i,j})$, Ω_i and R. The verification processes are similar to those above. As for insertion, $h(b_{i,j})$ has already been verified along with Ω_i; the client can safely compute the new h_{i_new} without additional verifications, see Fig. 4-a.

3. With RST structure, the client will then compute h'_i with $h'(b_{i,j})$, then compute R_{new} with Ω_i and h'_i and compare R_{new} with R'.

If all 3 verification passed, it means that the server has performed the update to all replicas honestly. The client will update the total block number n, then compute $\sigma'_{i,j}$ (the authenticators for $b'_{i,j}$) and store them on server.

The protocol for verification of updates is demonstrated in Fig. 5.

4.2.4 Challenge and Verification for Multi-replica Public Auditing

Within our top-down levelled setting, the verifier will need $H(b_{i,j})$ to verify the auditing equation as it is not stored in the MR-MHT. Here we discuss how to conduct verification on all replica blocks for a given set of indices in one go.

$GenChallenge(Acc, pk, sig_{AUTH})$: The third-party auditor TPA generates challenge message with the give accuracy Acc, and sends an authorization. For example, same as before, for a 99% accuracy, the verifier needs to verify 460 blocks out of a 1GB file. The challenge message is $\{sig_{AUTH}, i, v_{i,j}\}_{i\in I}$ where sig_{AUTH} is for authorization, I is the random set of indices chosen for verification, and $v_{i,j}$ are random numbers for integration of $b_{i,j}$.

$GenProof(pk, F, \Phi, chal)$: The cloud server will first verify sig_{AUTH}, same as in [17]. Then, it will compute $\sigma_j = \prod_{i\in I}\sigma_i^{v_i}$ and $\mu_j = \sum_j v_{i,j}b_{i,j}$ for every replica, and send $\{\mu_j, \sigma_j, \{H(b_{i,j}), h(b_{i,j}), \Omega_i\}_{i\in I}, sig\}$ back to TPA.

$Verify(pk, P)$: Since the verifier knows the structure of RSTs, it will compute R with $\{h(b_{i,j}),\Omega_i\}$ and verify sig for each ith chosen block. The verification process is similar as in section 4.2.3, with iterative triples and verification of $\xi_\pi = i - 1$ of . Also, it needs to verify the authenticity of $H(b_{i,j})$ by verifying if $h(b_{i,j}) = H\left(1||l(b_{i,j})||H(b_{i,j})\right)$, where $l(b_{i,j})$ can be inferred from $l(h_i)$ which equals level of the first node in Ω_i. For example, in Fig. 3, $c = 3$. When we know that $l(h_2) = 2$ from $l(h_1) = 2$ ($h_1$is the first node in Ω_2), we can easily derive $l(b_{2,1}) = 4, l(b_{2,2}) = 4, l(b_{2,3}) = 3$. If this verifications passed, TPA will trust the retrieved $H(b_{i,j})$ are genuine, then it can verify c replicas one by one by verifying the following c equations:

$$e(\sigma_j, g) = e\left(\prod_i H(b_{i,j})^{v_{i,j}}u^{\mu_j}, v\right), j \in [1, c]$$

If these equations holds then the verification will output 'ACCEPT', otherwise output 'REJECT'. The process is demonstrated in Fig. 6.

4.3 Discussions and Extensions

Since each replica block $b_{i,j}$ has its own authenticator $\sigma_{i,j}$, our scheme also supports single replica verification. The process will be similar to the verification in [31] with additional verification of $H(b_{i,j})$ and the index of $h(b_{i,j})$. Except for the rank verifications of ξ, ζ are now $\xi_\pi = (i-1)\cdot c + j - 1$ and $\zeta_\pi = (n - i + 1)\cdot c - j$. other details will be similar as the verifications described above.

In [23], the authors proposed a value s for trade-off of storage and communication overheads. In this strategy, every file block m_i is segmented into s segments m_{ik} (length of each segment equals the length of a block without s, typically 20bytes), and the authenticators are computed as $\sigma_i = \left(H(m_i)\prod_{k=1}^{s}u_k^{m_{ik}}\right)^{\alpha}$. In this case, the proof size has increased by $s\times$ because there will be multiple $\mu_k = \Sigma_i v_i m_{ik}$, instead of one, to be included in the proof. However, the storage overhead has decreased to $1/s$ as there is only one authenticator stored along with s sectors. As our scheme is also based on BLS signature, with same

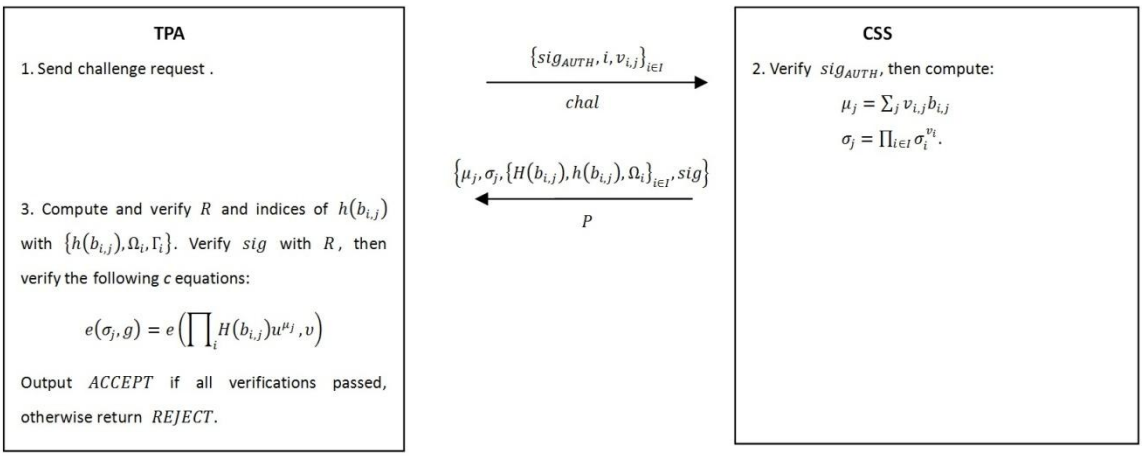

Fig. 6. Public auditing of all replicas at once

block segmentation strategy, the trade-off can easily be applied to our scheme to support dynamic data with multiple replicas. We will show our experimental results under different s values in Section 6.

Based on the segmented blocks, Liu, et, al. have investigated fine-grained updates for variable-sized file blocks with different segmentations and RMHT in [17]. If we extend MR-MHT to let the nodes to store the 'rank' information computed from different sizes of blocks, our scheme can also support fine-grained updates and enhance the scheme in [17] with efficient support for update of multiple replicas.

Wang et, al. have proposed a random masking technology for privacy protection against the third-party auditor [30]. In their scheme, the server will mask the proof μ (integrated blocks) with a random r and generate a new $\mu' = r + \gamma\mu$ so that TPA will not learn the users data from multiple challenging of the same set of blocks. In the multi-replica setting, the proof μ is computed based on replica blocks $b_{i,j}$ instead of the message blocks m_i. Therefore, in most scenarios it is not necessary to apply another masking from the server. Even TPA can infer $b_{i,j}$ from multiple challenges, it will not get any information of the user data m_i without knowing the transformation method, which is known only by the client, from $b_{i,j}$ to m_i. If there is any need to protect replica blocks against the TPA, our scheme can be extended with the same server-side padding strategy.

5 Security and Efficiency Analysis

As before, the security of our scheme is based on :
1. collision-resistance of the hash function,
2. difficulty of gap Diffie-Hellman problem, and
3. unforgeability of the chosen signature scheme.

5.1 Verifiable Multi-Replica Updates

Lemma 1. *With sig, RST structure, total number of blocks n and a given block index $i \leq n$, if a returned block-AAI combination $\{h_i, \Omega_i\}$ for an RST root passed the authentication, then either it is computed with the actual replica blocks, or the server has found a way to find collisions in the hash function H.*

Proof. The client will first infer $l(h_i)$, the level of h_i, from Ω_i. Let π be the number of tuples in Ω_i, then $l(h_i) = \pi$. If the a dishonest server does not have the ability to find arbitrary collisions of hash functions, it must select an existing node N and its corresponding AAI Ω_N in MR-MHT in order to let the client to compute R, thereby verify sig, through iterative hashing.

1. If N is not on the verification path of h_i, then either the server provides wrong level or rank values, which will lead to failure in computing the right R; or verification of both values of ξ_π and ζ_π will fail.
2. When the queried node is a left child node, choosing any hash value and its AAI from the verification path will let the verification process output the correct ξ_π, but not the correct ζ_π.
3. When the queried node is a right child node, choosing any hash value and its AAI from the verification path will let the verification process output the correct ζ_π, but not the correct ξ_π.

Therefore, except for finding hash collisions, the server must return the exact h_i in order to let all three values pass the verification. □

With the Lemma above, we can now describe the soundness and security of the update verification process in MuR-DPA through the following theorems.

Theorem 1. *If there is any fault to the new data content or index in the server execution of an update request $\{Type, i, \{b'_{i,j}\}\}$, the client verification will fail.*

Proof. According to Lemma 1, the RST root h_i and its AAI Ω_i returned by the server are the correct representative for the RST where $\{b_{i,j}\}$ resided, otherwise the verification of R will fail.

1. For insertions and modifications, if $b'_{i,j}$ was updated incorrectly, then h'_i , therefore R', will be computed incorrect due to the collision resistance of hash function H. According to the property of MHT, Ω_i stays the same throughout the update. As the client has the right $b'_{i,j}$ and Ω_i, the values h'_i and R' at client side will be correct. Therefore, the verification will fail.

2. For deletions, the returned $h(b'_{i,j})$ will be incorrect once there is any fault in this update. As $h(b'_{i,j})$ is included in the $\Omega(b_{i,j})$, the client will identify if $h(b'_{i,j})$ is

incorrect.

Therefore, through the verification, the client will be able to detect any fault caused by accident or dishonest behaviours in the update. □

This concludes the proof that the MuR-DPA scheme can support public auditing of dynamic data without cheated by a dishonest server. As for efficiency, the AAI Ω_i will be taking the majority of data transfer because it is composed of $\log(n)$ hash values and rank/level information for each update. For updating of multiple replicas (which is a must for cloud storage with multiple replicas), only one, instead of c AAIs, is needed to be transferred for verification of c replica blocks. Therefore, the more replica there is, the more efficiency advantage our scheme would have.

5.2 All-at-once Multi-Replica Verification

Same as verifiable updates, there is need for verification of Ω_i.

Theorem 3. *In MuR-DPA scheme, If integrity of any replica $b_{i,j}$ of the i-th block was breached, the server cannot build a response $\{\mu_j, \sigma_j, \{H(b_{i,j}), h(b_{i,j}), \Omega_i\}_{i\in I}, sig\}$ that can successfully pass the verification, unless any of the 3 assumptions at the beginning of this section fails to hold.*

Proof. As the structure of RST is known by the verifier, the verifier will be able to re-build the RST under $h(i)$, thereby compute $h(i)$ based on $h(b_{i,j})$. With Lemma 1, the authenticity of $\{h_i, \Omega_i\}$ can be verified via sig, i and n. Therefore, if $h(b_{i,j}), l(b_{i,j}), r(b_{i,j})$ are not all correct, then $h(i)$ will be incorrect; with Ω_i, the verification for R will fail. Because $h(b_{i,j})$ was computed with $r(b_{i,j}), l(b_{i,j})$ and $H(b_{i,j})$, if all these 3 values are correct, then the returned $H(b_{i,j})$ must be correct, otherwise the client will fail to verify the equation $h(b_{i,j}) = H\left(r(b_{i,j})||l(b_{i,j})||H(b_{i,j})\right)$. Therefore, our design can make sure the returned $H(b_{i,j})$ are indeed the hash values of the designated replicas for the ith block. On the other hand, the soundness and security of verification equation $e(\sigma_j, g) = e(\prod_i H(b_{i,j})^{v_{i,j}} u^{\mu_j}, v)$ itself has already been proven in [23] and [31]. Therefore, any integrity breach will be identified with MuR-DPA. □

The proof above is based on the assumption that the verifier knows the structure of RST. In fact, even when the RST structure was unknown to the verifier, the verification for all replicas may still be resilient to dishonest servers as exchanging the orders of replicas under an RST does not affect the verification. We leave this problem as future work.

As a drawback, MR-MHT introduced more levels (depth of RSTs) than each MHT in SiR-DPA to store replica blocks. Therefore, the verification cost for one replica in MuR-DPA will be slightly larger than in SiR-PA. However, as replica number is small (usually less than 10), the depth of RSTs are only less than 4 levels. Therefore, no significant overhead for the client to verify a single replica. Details will be discussed in the next section.

TABLE 1
PRICE OF DYNAMISM - COMMUNICATION OVERHEAD FOR VERIFYING UPDATES OF HALF BLOCKS IN A 1GB FILE

s (number of sectors per block)	Data Updated (MB)	Total Server Response for Verification (MB)
1	512/1024	19.507
5	512/1024	3,625
10	512/1024	1,743
20	512/1024	837
50	512/1024	321
100	512/1024	154

6 EXPERIMENTAL RESULTS AND ANALYSIS

6.1 Experimental Environment

We conducted our experiments on U-Cloud -- a cloud computing environment located in University of Technology, Sydney (UTS). The computing facilities of this system are located in several labs in the Faculty of Engineering and IT, UTS. On top of hardware and Linux OS, We installed KVM Hypervisor [3] which virtualizes the infrastructure and allows it to provide unified computing and storage resources. Upon virtualized data centers, Hadoop [2] is installed to facilitate the MapReduce programming model and distributed file system. Moreover, we installed OpenStack open source cloud platform [4] which is responsible for global management, resource scheduling, task distribution and interaction with users. The structure of U-Cloud is demonstrated in Fig. 7.

6.2 Experimental Evaluations

We compare our new scheme, MuR-DPA, against the direct extension of the existing scheme in [31] with tags of each replica indexed in separate MHTs and MHTs are with levels and ranks for index authentication. We name this scheme as SiR-DPA - Dynamic Public Auditing with Separately-indexed Replicas. As the computation time is not the primary concern in public auditing schemes, we will mainly measure communication and storage costs. All experiments are conducted on an 1GB randomly generated dataset and its replicas computed as $b_{i,j} = m_{i,j} + ri,j$, on a virtual server launched in U-Cloud with 4 cores and 16GB RAM. BLS parameters are chosen with 80-bit security.

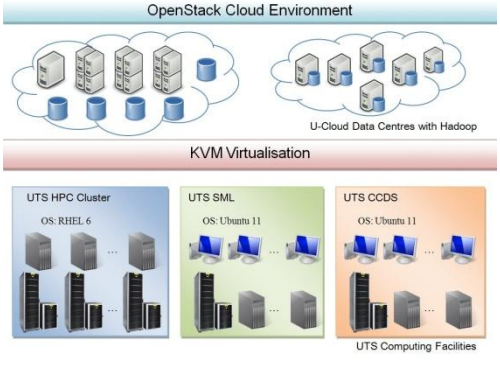

Fig. 7. U-Cloud environment

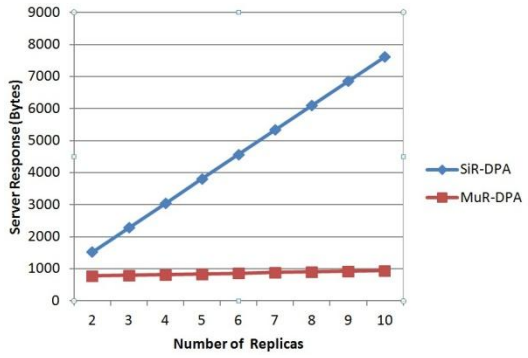

Fig. 8. Length of server response for one verifiable modification/insertion of one block

We first measured the communication cost for verification of updates. Table 1 shows the total communication overhead for update verification of only one replica, where overheads of SiR-PA and MuR-PA are the same. The testing dataset is 1 GB and we are updating half of the blocks with 512MB new content in total; with adjusting parameter s. Communication overhead for update verification in the protocol in [14] and the MHT-based scheme in [31] will be similar to our SiR-DPA setting, as the communication complexities in MHT and RASL are both $O(\log n)$ with high propabilitiy (whp). Note that in this experiment, there is only one update for each block for all modifications. Under this setting, we can see that this overhead is always a large burden. Even for a large $s = 100$, there's still 154MB verification data needed to be transferred from the server for updating of 512MB data. Although the communication cost will decrease for a larger block size (because the number of blocks will be smaller), it may take several update processes to update half of its content, where the communication cost will increase beyond the amount in Table 1. To make things worse, with multiple replicas, SiR-DPA scheme will multiply this communication cost, which has to be avoided if possible, given the fact that cloud service providers always keep multiple replicas for storage services.

Second, we tested the communication cost for updates with different numbers of replicas and different sizes of blocks. Results are depicted in Fig. 8 and 9. From Fig. 8, we can see that the length of server response for modification and insertion has been greatly reduced when there are multiple replicas, which means the server's crucial downlink bandwidth will be greatly reduced. The more replicas the dataset has, the more advantageous the MuR-DPA scheme is. Overheads for deletions will be similar as there is only one more hash value to be included in server response, thus the comparison is omitted here. The total communication overheads for verification of updates to dataset with multiple replicas are also tested. For block insertion and modification, the new data block need to be uploaded. Therefore, for a larger s, (i.e. a larger block size), the total communication cost will rise. For block deletion, since there is no new data block, nothing needs to be uploaded. Therefore, the total communication for a single deletion stays unchanged with different s values. Either way, for s=1 and s=10, our results show that communication overheads of verification of updates in MuR-DPA always has significant advantage compared to SiR-DPA.

Third, we tested extra storage overhead for dynamic public auditability, as well as communication overhead for auditing of multiple replicas at once. Although the total number of authenticators stayed the same, now it is only one MHT (although with more levels) as opposed to c MHTs in SiR-DPA. We can infer from Fig. 10 that the extra storage cost is reduced by a great percentage when there are multiple replicas stored in cloud. Communication overheads for verification of multiple replicas at once are depicted in Fig 11. We can see that the more replicas the server stored, the more advantage MuR-DPA scheme has against SiR-DPA scheme. We have also noticed that with the growth of replica number, the communication overhead for verifying all replicas with MuR-DPA scheme is close to verifying a single replica, while the overhead of SiR-DPA grows in a much faster pace. For example, when $s = 1, c = 5$, verifying all 5 replicas with MuR-DPA takes 26.8% more communication than verifying only 1 replica, while this percentage for SiR-DPA is 398.8%. Therefore, the MuR-DPA scheme is not only useful for verification of dynamic data, but also especially useful for auditing of important datasets to ensure all their replicas are intact.

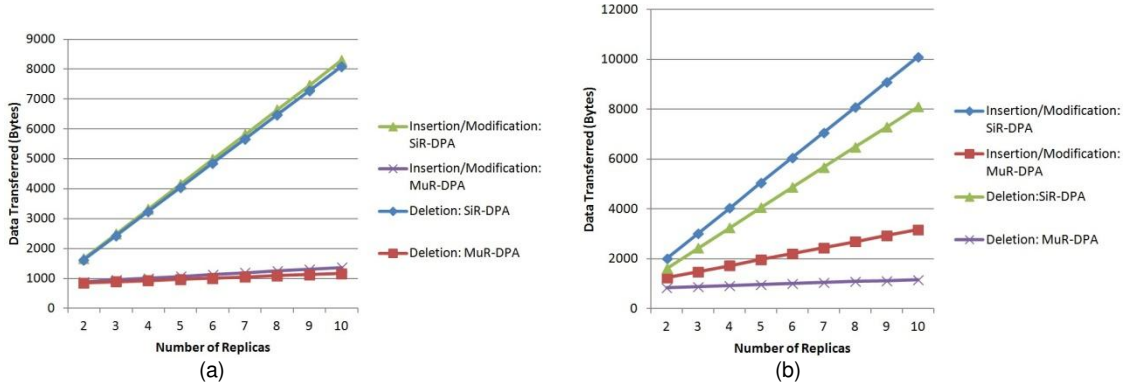

Fig. 9. Total communication for one verifiable update of one block when (a) $s = 1$; (b) $s = 10$

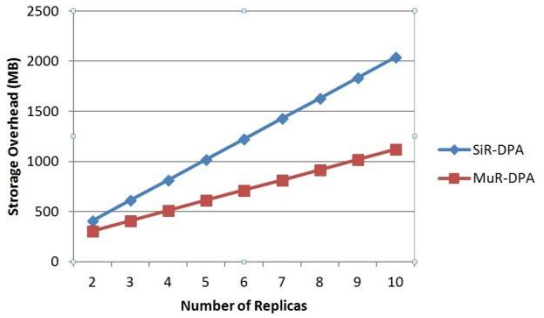

Fig. 10. Extra storage overhead at server side for support of public auditability and data dynamics

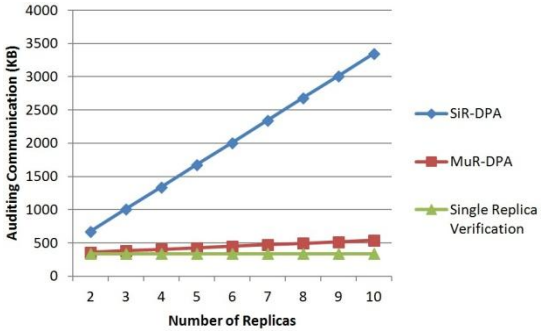

Fig. 11-a. Total communication for auditing of all replicas when s = 1

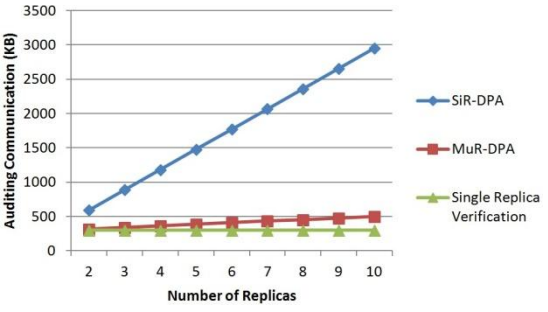

Fig. 11-b. Total communication for auditing of all replicas when s = 10

We also tested the communication cost for one replica, under different s value. As analysed in section 5, our scheme will constantly incur more communication overhead because of the extended RSTs. However, as can be seen from Fig. 12, the extra communication overhead is small. Even for an exaggerated case where $s = 1{,}000$ and $c = 8$, the extra communication for verification of one replica in MuR-DPA scheme is only 15.3% compared to SiR-DPA scheme. For a more common choice of 4 replicas and $s = 10$, this percentage is only 8.1%. Given that the MuR-DPA scheme has much less communication cost for verification of all replicas at once as well as verification of updates, we would consider this an advantageous trade-off.

7 Conclusions

In this paper, we presented a novel public auditing scheme named MuR-DPA. The new scheme incorporated a novel authenticated data structure based on the Merkle hash tree, which we name as MR-MHT. For support of full dynamic data updates, authentication of block indices

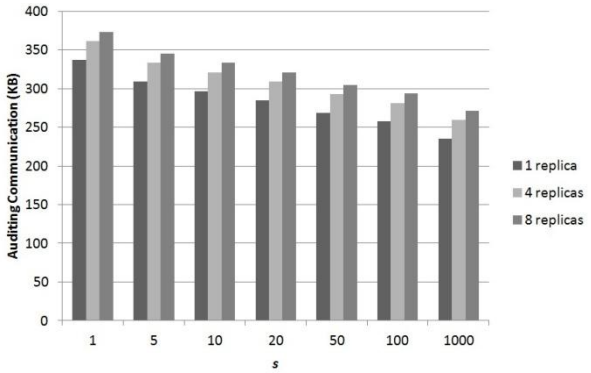

Fig. 12. Communication for auditing of 1 chosen replica for a data-set with 1, 4 and 8 total replicas with different s

and efficient verification of updates for multiple replicas at the same time, the level values of nodes in MR-MHT are generated in a top-down order, and all replica blocks for each data block are organized into a same replica sub-tree. Compared to existing integrity verification and public auditing schemes, theoretical analysis and experimental results have shown that the MuR-DPA scheme can not only incur much less communication overhead for both update and verification of datasets with multiple replicas, but also provide enhanced security against dishonest cloud service providers.

Acknowledgment

This work is supported in part by ARC LP0990393.

References

[1] Available: http://aws.amazon.com/apac/awssummit-au/, accessed on 25 March, 2013.

[2] *Hadoop MapReduce*. Available: http://hadoop.apache.org, accessed on 25 March, 2013.

[3] *KVM Hypervisor*. Available: www.linux-kvm.org/, accessed on 25 March, 2013.

[4] *OpenStack Open Source Cloud Software*. Available: http://openstack.org/, accessed on 25 March, 2013.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, pp. 50-58, 2010.

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote Data Checking Using Provable Data Possession," *ACM Transactions on Information and System Security*, vol. 14, p. Article 12, 2011.

[7] G. Ateniese, R. B. Johns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, 2007, pp. 598-609

[8] G. Ateniese, S. Kamara, and J. Katz, "Proofs of Storage from Homomorphic Identification Protocols," in *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '09)*, Tokyo, Japan, 2009, pp. 319 - 333.

[9] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," in *Proceedings of the 4th*

International Conference on Security and Privacy in Communication Netowrks (SecureComm '08), İstanbul, Turkey, 2008, pp. 1-10.

[10] D. Boneh, H. Shacham, and B. Lynn, "Short Signatures from the Weil Pairing," *Journal of Cryptology,* vol. 17, pp. 297-319, 2004.

[11] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems,* vol. 25, pp. 599-616, 2009.

[12] D. Cash, A. Küpçü, and D. Wichs, "Dynamic Proofs of Retrievability via Oblivious RAM," in *Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '13)*, Athens, Greece, 2013, pp. 279-295.

[13] R. Curtmola, O. Khan, R. C. Burns, and G. Ateniese:, "MR-PDP: Multiple-Replica Provable Data Possession. ," in *Proceedings of the 28th IEEE International Conference on Distributed Computing Systems (ICDCS '08)*, Beijing, China, 2008, pp. 411-420.

[14] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*, Chicago, USA, 2009, pp. 213-222.

[15] Y. He, S. Barman, and J. F. Naughton, "Preventing Equivalence Attacks in Updated, Anonymized Data," in *Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE '11)*, 2011, pp. 529-540.

[16] A. Juels and J. B. S. Kaliski, "PORs: Proofs of Retrievability for Large Files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, Alexandria, USA, 2007, pp. 584-597.

[17] C. Liu, J. Chen, L. T. Yang, X. Zhang, C. Yang, R. Ranjan, and K. Ramamohanarao, "Authorized Public Auditing of Dynamic Big Data Storage on Cloud with Efficient Verifiable Fine-grained Updates," *IEEE Transactions on Parallel and Distributed Systems,* 2013.

[18] C. Liu, X. Zhang, C. Yang, and J. Chen, "CCBKE - Session Key Negotiation for Fast and Secure Scheduling of Scientific Applications in Cloud Computing," *Future Generation Computer Systems,* 2012.

[19] T. Mather, S. Kumaraswamy, and S. Latif, *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. Sebastopol: O'Reilly Media, 2009.

[20] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Proceedings of A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology (CRYPTO '87)*, 1987, pp. 369-378.

[21] E. Naone. *What Twitter Learns from All Those Tweets*. Available: http://www.technologyreview.com/view/420968/what-twitter-learns-from-all-those-tweets/, accessed on 25 March, 2013.

[22] S. E. Schmidt. *Security and Privacy in the AWS Cloud*. Available: http://aws.amazon.com/apac/awssummit-au/, accessed on 25 March, 2013.

[23] H. Shacham and B. Waters, "Compact Proofs of Retrievability," in *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '08)*, 2008, pp. 90 - 107

[24] E. Shi, E. Stefanov, and C. Papamanthou, "Practical Dynamic Proofs of Retrievability," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*, 2013, pp. 325-336.

[25] E. Stefanov, M. v. Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An Extremely Simple Oblivious RAM Protocol," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*, 2013, pp. 299-310.

[26] S. Subashini and V. Kavitha, "A Survey on Security Issues in Service Delivery Models of Cloud Computing," *Journal of Network and Computer Applications,* vol. 34, pp. 1-11, 2010.

[27] B. Wang, S. S. M. Chow, M. Li, and H. Li, "Storing Shared Data on the Cloud via Security-Mediator," in *33rd IEEE International Conference on Distributed Computing Systems (ICDCS '13)*, Philadelphia, USA, 2013.

[28] B. Wang, B. Li, and H. Li, "Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing (CLOUD '12)*, Hawaii, USA, 2012, pp. 295-302.

[29] B. Wang, B. Li, and H. Li, "Public Auditing for Shared Data with Efficient User Revocation in the Cloud," in *Proceedings of the 32nd Annual IEEE International Conference on Computer Communications (INFOCOM'13)*, Turin, Italy, 2013, pp. 2904-2912.

[30] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," in *Proceedings of the 29th Annual IEEE International Conference on Computer Communications (INFOCOM'10)*, San Diego, USA, 2010, pp. 1 - 9.

[31] 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, pp. 847 - 859, 2011.

[32] P. Williams, R. Sion, and A. Tomescu, "PrivateFS: A Parallel Oblivious File System," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, 2012, pp. 977-988.

[33] J. Yao, S. Chen, S. Nepal, D. Levy, and J. Zic, "TrustStore: Making Amazon S3 Trustworthy with Services Composition," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID '10)*, Melbourne, Australia, 2010, pp. 600-605.

[34] X. Zhang, C. Liu, S. Nepal, S. Panley, and J. Chen, "A Privacy Leakage Upper-bound Constraint based Approach for Cost-effective Privacy Preserving of Intermediate Datasets in Cloud," *IEEE Transactions on Parallel and Distributed Systems,* 2012.

[35] Y. Zhang and M. Blanton, "Efficient Dynamic Provable Possession of Remote Data via Update Trees," 2012.

[36] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative Provable Data Possession for Integrity Verification in Multi-Cloud Storage," *IEEE Transactions on Parallel and Distributed Systems,* vol. 23, pp. 2231-2244, 2012.

[37] D. Zissis and D. Lekkas, "Addressing Cloud Computing Security Issues," *Future Generation Computer Systems,* vol. 28, pp. 583-592, 2011.