Authorized Public Auditing of Dynamic Big Data Storage on Cloud with Efficient Verifiable Fine-Grained Updates

Chang Liu, Jinjun Chen, *Senior Member, IEEE*, Laurence T. Yang, *Member, IEEE*, Xuyun Zhang, Chi Yang, Rajiv Ranjan, and Ramamohanarao Kotagiri

Abstract—Cloud computing opens a new era in IT as it can provide various elastic and scalable IT services in a pay-as-you-go fashion, where its users can reduce the huge capital investments in their own IT infrastructure. In this philosophy, users of cloud storage services no longer physically maintain direct control over their data, which makes data security one of the major concerns of using cloud. Existing research work already allows data integrity to be verified without possession of the actual data file. When the verification is done by a trusted third party, this verification process is also called data auditing, and this third party is called an auditor. However, such schemes in existence suffer from several common drawbacks. First, a necessary authorization/authentication process is missing between the auditor and cloud service provider, i.e., anyone can challenge the cloud service provider for a proof of integrity of certain file, which potentially puts the quality of the so-called 'auditing-as-a-service' at risk; Second, although some of the recent work based on BLS signature can already support fully dynamic data updates over fixed-size data blocks, they only support updates with fixed-sized blocks as basic unit, which we call coarse-grained updates. As a result, every small update will cause re-computation and updating of the authenticator for an entire file block, which in turn causes higher storage and communication overheads. In this paper, we provide a formal analysis for possible types of fine-grained data updates and propose a scheme that can fully support authorized auditing and fine-grained update requests. Based on our scheme, we also propose an enhancement that can dramatically reduce communication overheads for verifying small updates. Theoretical analysis and experimental results demonstrate that our scheme can offer not only enhanced security and flexibility, but also significantly lower overhead for big data applications with a large number of frequent small updates, such as applications in social media and business transactions.

Index Terms—Cloud computing, big data, data security, provable data possession, authorized auditing, fine-grained dynamic data update

1 Introduction

Cloud computing is being intensively referred to as one of the most influential innovations in information technology in recent years [1], [2]. With resource virtualization, cloud can deliver computing resources and services in a pay-as-you-go mode, which is envisioned to become as convenient to use similar to daily-life utilities such as electricity, gas, water and telephone in the near future [1]. These computing services can be categorized into Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) [3]. Many international IT corporations now offer powerful public cloud services to users on a scale from individual to enterprise all over the world; examples are Amazon AWS, Microsoft Azure, and IBM SmartCloud.

Although current development and proliferation of cloud computing is rapid, debates and hesitations on the usage of cloud still exist. Data security/privacy is one of the major concerns in the adoption of cloud computing [3], [4], [5]. Compared to conventional systems, users will lose their direct control over their data. In this paper, we will investigate the problem of integrity verification for big data storage in cloud. This problem can also be called data auditing [6], [7] when the verification is conducted by a trusted third party. From cloud users' perspective, it may also be called 'auditing-as-a-service'. To date, extensive research is carried out to address this problem [6], [7], [8], [9], [10], [11], [12], [13], [14], [15]. In a remote verification scheme, the cloud storage server (CSS) cannot provide a valid integrity proof of a given proportion of data to a verifier unless all this data is intact. To ensure integrity of user data stored on cloud service provider, this support is of no less importance than any data protection mechanism deployed by the cloud service provider (CSP) [16], no matter how secure they seem to be, in that it will provide the verifier a piece of direct, trustworthy and real-timed intelligence of the integrity of the cloud user's data through a challenge request. It is especially recommended that data auditing is to be conducted on a regular basis for the

- *C. Liu is with the School of Comput. Sci. and Tech., Huazhong Uni. of Sci. and Tech., China, and also with the Faculty of Eng. and IT, Uni. of Tech., Sydney, Australia. E-mail: changliu.it@gmail.com.*
- *J. Chen, X. Zhang, and C.Yang are with the Faculty of Eng. and IT, Uni. of Tech., Sydney, Australia. E-mail: {jinjun.chen, xyzhanggz, chiyangit}@gmail.com.*
- *L.T. Yang is with the School of Comput. Sci. and Tech., Huazhong Uni. of Sci. and Tech., China, and also with the Dept. of Comput. Sci., St. Francis Xavier Uni., Canada. E-mail: ltyang@stfx.ca.*
- *R. Ranjan is with CSIRO Computational Informatics Division, Australia. E-mail: rranjans@gmail.com.*
- *R. Kotagiri is with the Dept. of Comput. and Information Systems, The Uni. of Melbourne, Australia. E-mail: kotagiri@unimelb.edu.au.*

Manuscript received 21 May 2013; revised 23 July 2013; accepted 24 July 2013. Date of publication 4 Aug. 2013; date of current version 13 Aug. 2014. Recommended for acceptance by Y. Xiang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.191

users who have high-level security demands over their data.

Although existing data auditing schemes already have various properties (see Section 2), potential risks and inefficiency such as security risks in unauthorized auditing requests and inefficiency in processing small updates still exist. In this paper, we will focus on better support for small dynamic updates, which benefits the scalability and efficiency of a cloud storage server. To achieve this, our scheme utilizes a flexible data segmentation strategy and a ranked Merkle hash tree (RMHT). Meanwhile, we will address a potential security problem in supporting public verifiability to make the scheme more secure and robust, which is achieved by adding an additional authorization process among the three participating parties of client, CSS and a third-party auditor (TPA).

Research contributions of this paper can be summarized as follows:

1. For the first time, we formally analyze different types of fine-grained dynamic data update requests on variable-sized file blocks in a single dataset. To the best of our knowledge, we are the first to propose a public auditing scheme based on BLS signature and Merkle hash tree (MHT) that can support fine-grained update requests. Compared to existing schemes, our scheme supports updates with a size that is not restricted by the size of file blocks, thereby offers extra flexibility and scalability compared to existing schemes.
2. For better security, our scheme incorporates an additional authorization process with the aim of eliminating threats of unauthorized audit challenges from malicious or pretended third-party auditors, which we term as 'authorized auditing'.
3. We investigate how to improve the efficiency in verifying frequent small updates which exist in many popular cloud and big data contexts such as social media. Accordingly, we propose a further enhancement for our scheme to make it more suitable for this situation than existing schemes. Compared to existing schemes, both theoretical analysis and experimental results demonstrate that our modified scheme can significantly lower communication overheads.

For the convenience of the readers, we list some frequently-used acronyms in Appendix 1 which is available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.191.

1.1 Paper Organization

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 provides motivating examples as well as a detailed analysis of our research problem. Section 4 provides a description of our proposed scheme in detail, with also a detailed analysis of fine-grained update requests and how they can be supported. Section 5 provides security analysis for our design. Section 6 provides experimental results. Section 7 concludes our research and points out future work.

2 Related Work

Compared to traditional systems, scalability and elasticity are key advantages of cloud [1], [2], [3]. 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 [6], [8], [12], [17]. In this paper, we will focus on small and frequent data updates, which is important because these updates exist in many cloud applications such as business transactions and online social networks (e.g. Twitter [18]). Cloud users may also need to split big datasets into smaller datasets and store them in different physical servers for reliability, privacy-preserving or efficient processing purposes.

Among the most pressing problems related to cloud is data security/privacy [4], [5], [19]. It has been one of the most frequently raised concerns [5], [20]. There is a lot of work trying to enhance cloud data security/privacy with technological approaches on CSP side, such as [21], [22]. As discussed in Section 1, they are of equal importance as our focus of external verifications.

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.* [14]. 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) [10]. 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 pre-computed file tags which they call homomorphic verifiable tags (HVTs) or homomorphic linear authenticators (HLAs). Work by Shacham, *et al.* [15] provided an improved POR model with stateless verification. They also proposed a MAC-based private verification scheme and the first public verification scheme in the literature that based on BLS signature scheme [23]. In their second 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 is a desired benefit for a POR scheme. They also proved the security of both their schemes and the PDP scheme by Ateniese, *et al.* [9], [10]. From then on, the concepts of PDP and POR were in fact unified under this new compact POR model. Ateniese, *et al.* extended their scheme for enhanced scalability [8], but only partial data dynamics and a predefined number of challenges is supported. In 2009, Erway, *et al.* proposed the first PDP scheme based on skip list that can support full dynamic data updates [12]. However, public auditability and variable-sized file blocks are not supported by default. Wang, *et al.* [6] proposed a scheme based on BLS signature that can support public auditing (especially from a third-party auditor, TPA) and full data dynamics, which is one of the latest works on public data auditing with dynamics support. However, their scheme lacks support for fine-grained update and authorized auditing which are the main focuses of our work. Latest work by Wang *et al.* [7] added a random masking technology on top of [6] to ensure the TPA cannot infer the raw data file from a series of integrity

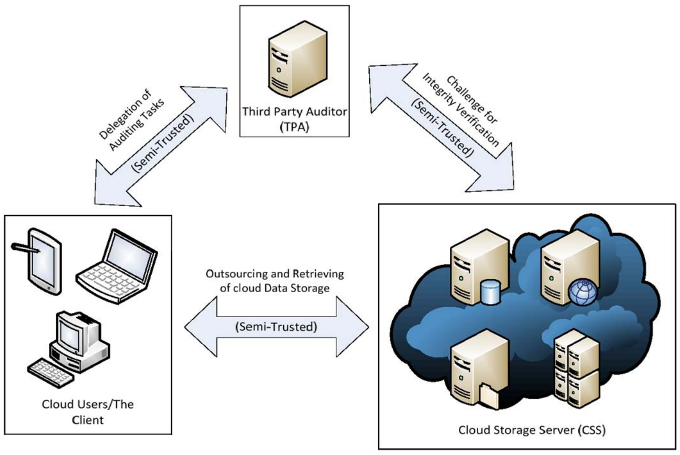
Third Party Auditor (TPA)
Delegation of Auditing Tasks
(Semi-Trusted)
Challenge for Integrity Verification
(Semi-Trusted)
Outsourcing and Retrieving of cloud Data Storage
(Semi-Trusted)
Cloud Users/The Client
Cloud Storage Server (CSS)

Fig. 1. Relationship between the participating parties in a public auditing scheme.

proofs. In their scheme, they also incorporated a strategy first proposed in [15] to segment file blocks into multiple ‘sectors’. However, the use of this strategy was limited to trading-off storage cost with communication cost.

Other lines of research in this area include the work of Ateniese, *et al.* [24] on how to transform a mutual identification protocol to a PDP scheme; scheme by Zhu, *et al.* [13] that allows different service providers in a hybrid cloud to cooperatively prove data integrity to data owner; and the MR-PDP Scheme based on PDP [10] proposed by Curtmola, *et al.* [11] that can efficiently prove the integrity of multiple replicas along with the original data file.

3 Motivating Examples and Problem Analysis

3.1 Motivating Examples

Cost-efficiency brought by elasticity is one of the most important reasons why cloud is being widely adopted. For example, Vodafone Australia is currently using Amazon cloud to provide their users with mobile online-video-watching services. According to their statistics, the number of video requests per second (RPS) can reach an average of over 700 during less than 10 percent of the time such as Friday nights and public holidays, compared to a mere 70 in average in the rest 90 percent of the time. The variation in demand is more than 9 times [3]. Without cloud computing, Vodafone cannot avoid purchasing computing facilities that can process 700 RPS, but it will be a total waste for most of the time. This is where cloud computing can save a significant amount of investments—cloud’s elasticity allows the user-purchased computation capacity to scale up or down on-the-fly at any time. Therefore, user requests can be fulfilled without wasting investments in computational powers. Other 2 large companies who own *news.com.au* and *realestate.com.au*, respectively, are using Amazon cloud for the same reason [3]. We can see through these cases that scalability and elasticity, thereby the capability and efficiency in supporting data dynamics, are of extreme importance in cloud computing.

Many big data applications will keep user data stored on the cloud for small-sized but very frequent updates. A most typical example is Twitter, where each tweet is restricted to 140 characters long (which equals 140 bytes in ASCII code). They can add up to a total of 12 terabytes of data per day [18]. Storage of transaction records in banking or securities markets is a similar and more security-heavy example. Moreover, cloud users may need to split large-scale datasets into smaller chunks before uploading to the cloud for privacy-preserving [17] or efficient scheduling [19]. In this regard, efficiency in processing small updates is always essential in big data applications.

To better support scalability and elasticity of cloud computing, some recent public data auditing schemes do support data dynamics. However, types of updates supported are limited. Therefore previous schemes may not be suitable for some practical scenarios. Besides, there is a potential security threat in the existing schemes. We will discuss these problems in detail in the next Section 3.2.

3.2 Problem Analysis

3.2.1 Roles of the Participating Parties

Most PDP and POR schemes can support public data verification. In such schemes, there are three participating parties: client, CSS and TPA. Relationships between the three parties are shown in Fig. 1. In brief, both CSS and TPA are only semi-trusted to the client. In the old model, the challenge message is very simple so that everyone can send a challenge to CSS for the proof of a certain set of file blocks, which can enable malicious exploits in practice. First, a malicious party can launch distributed denial-of-service (DDOS) attacks by sending multiple challenges from multiple clients at a time to cause additional overhead on CSS and congestion to its network connections, thereby causing degeneration of service qualities. Second, an adversary may get privacy-sensitive information from the integrity proofs returned by CSS. By challenging the CSS multiple times, an adversary can either get considerable information about user data (due to the fact that returned integrity proofs are computed with client-selected data blocks), or gather statistical information about cloud service status. To this end, traditional PDP models cannot quite meet the security requirements of ‘auditing-as-a-service’, even though they support public verifiability.

3.2.2 Verifiable Fine-Grained Dynamic Data Operations

Some of the existing public auditing schemes can already support full data dynamics [6], [7], [12]. In their models, only insertions, deletions and modifications on fixed-sized blocks are discussed. Particularly, in BLS-signature-based schemes [6], [7], [13], [15] with 80-bit security, size of each data block is either restricted by the 160-bit prime group order p, as each block is segmented into a fixed number of 160-bit sectors. This design is inherently unsuitable to support variable-sized blocks, despite their remarkable advantage of shorter integrity proofs. In fact, as described in Section 2, existing schemes can only support insertion, deletion or modification of one or multiple fixed-sized blocks, which we call ‘coarse-grained’ updates.

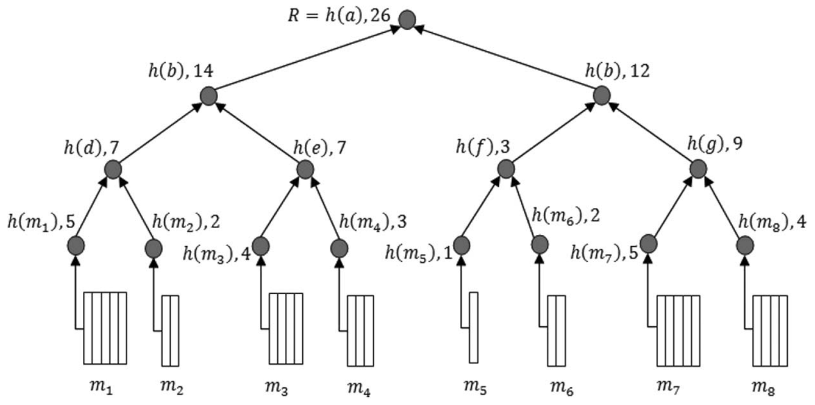

Fig. 2. Example of a rank-based Merkle hash tree (RMHT).

Although support for coarse-grained updates can provide an integrity verification scheme with basic scalability, data updating operations in practice can always be more complicated. For example, the verifiable update process introduced in [6], [12] cannot handle deletions or modifications in a size lesser than a block. For insertions, there is a simple extension that enables insertion of an arbitrary-sized dataset—CSS can always create a new block (or several blocks) for every insertion. However, when there are a large number of small upgrades (especially insertions), the amount of wasted storage will be huge. For example, in [6], [12] the recommended size for a data block is 16k bytes. For each insertion of a 140-byte Twitter message, more than 99 percent of the newly allocated storage is wasted—they cannot be reused until the block is deleted. These problems can all be resolved if fine-grained data updates are supported. According to this observation, supporting of fine-grained updates can bring not only additional flexibility, but also improved efficiency.

Our model assumes the following:

Assumption 1. *CSS will honestly answer all data queries to its clients. In other words, if a user asks to retrieve a certain piece of her data stored on CSS, CSS will not try to cheat her with an incorrect answer.*

This assumption—reliability—should be a basic service quality guarantee for cloud storage services.

PDP and POR are different models with similar goals. One main difference is that the file is encoded with error-correction code in the POR model, but not in the PDP model [6]. As in [6], [7], we will not restrict our work to either of the models.

4 The Proposed Scheme

Some common notations are introduced in Appendix A.

4.1 Preliminaries

4.1.1 Bilinear Map

Assume a group G is a gap Diffie-Hellman (GDH) group with prime order p. A bilinear map is a map constructed as $e : G \times G \rightarrow G_T$ where G_T is a multiplicative cyclic group with prime order. A useful e should have the following properties: bilinearity—$\forall m, n \in G \Rightarrow e(m^a, n^b) = e(m, n)^{ab}$; non-degeneracy—$\forall m \in G,\ m \neq 0 \Rightarrow e(m, m) \neq 1$; and computability—$e$ should be efficiently computable. For simplicity, we will use this symmetric bilinear map in our scheme description. Alternatively, the more efficient asymmetric bilinear map $e : G_1 \times G_2 \rightarrow G_T$ may also be applied, as was pointed out in [23].

4.1.2 Ranked Merkle Hash Tree (RMHT)

The Merkle Hash Tree (MHT) [25] has been intensively studied in the past. In this paper we utilize an extended MHT with ranks which we named RMHT. 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 2 child nodes. Information contained in one node N in an RMHT T is represented as $\{\mathcal{H}, r_N\}$ where $\mathcal{H}$ is a hash value and r_N is the rank of this node. T is constructed as follows. For a leaf node LN based on a message m_i, we have $\mathcal{H} = h(m_i)$, $r_{LN} = s_i$; A parent node of $N_1 = \{\mathcal{H}_1, r_{N1}\}$ and $N_2 = \{\mathcal{H}_2, r_{N2}\}$ is constructed as $N_P = \{h(\mathcal{H}_1 \| \mathcal{H}_2), (r_{N1} + r_{N2})\}$ where $\|$ is a concatenation operator. A leaf node m_i's 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 $\{m_i, \Omega_i\}$. For example, for the RMHT demonstrated in Fig. 2, m_1's AAI $\Omega_1 = \{h(m_2), h(e), h(d)\}$. According to the property of RMHT, we know that the number of hash values included in Ω_i equals the depth of m_i in T.

4.2 Framework and Definitions

We first define the following block-level fine-grained update operations:

Definition 1 (Types of Block-Level Operations in Fine-Grained Updates). *Block-level operations in fine-grained dynamic data updates may contain the following 6 types of operations: partial modification $\mathcal{PM}$—a consecutive part of a certain block needs to be updated; whole-block modification $\mathcal{M}$—a whole block needs to be replaced by a new set of data; block deletion $\mathcal{D}$—a whole block needs to be deleted from the tree structure; block insertion $\mathcal{J}$—a whole block needs to be created on the tree structure to contain newly inserted data;*

and block splitting $\mathcal{SP}$—a part of data in a block needs to be taken out to form a new block to be inserted next to it.[1]

Framework of public auditing scheme with data dynamics support is consisted of a series of algorithms. Similar to [12], the algorithms in our framework are: *Keygen*, *FilePreProc*, *Challenge*, *Verify*, *Genproof*, *PerformUpdate* and *VerifyUpdate*. Detailed definitions and descriptions can be found in Appendix B.

4.3 Our Scheme

We now describe our proposed scheme in the aim of supporting variable-sized data blocks, authorized third-party auditing and fine-grained dynamic data updates.

4.3.1 Overview

Our scheme is described in three parts:

1. Setup: the client will generate keying materials via *KeyGen* and *FileProc*, then upload the data to CSS. Different from previous schemes, the client will store a RMHT instead of a MHT as metadata. Moreover, the client will authorize the TPA by sharing a value sig_{AUTH}.
2. Verifiable Data Updating: the CSS performs the client's fine-grained update requests via *PerformUpdate*, then the client runs *VerifyUpdate* to check whether CSS has performed the updates on both the data blocks and their corresponding authenticators (used for auditing) honestly.
3. Challenge, Proof Generation and Verification: Describes how the integrity of the data stored on CSS is verified by TPA via *GenChallenge*, *GenProof* and *Verify*.

We now describe our scheme in detail as follows.

4.3.2 Setup

This phase is similar to the existing BLS-based schemes except for the segmentation of file blocks. Let $e : G \times G \rightarrow G_T$ be a bilinear map defined in Section 4.1, where G is a GDH group supported by $\mathbb{Z}_p$[2]. $H : (0,1)^* \rightarrow G$ is a collision-resistant hash function, and h is another cryptographic hash function.

After all parties have finished negotiating the fundamental parameters above, the client runs the following algorithms:

$KeyGen(1^k)$: The client generates a secret value $\alpha \in \mathbb{Z}_p$ and a generator g of G, then compute $v = g^\alpha$. A 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. For simplicity, in our settings, we use the same key pair for signatures, i.e., $ssk = \alpha, spk = \{v, g\}$.

$FilePreProc(F, sk, SegReq)$: According to the preemptively determined segmentation requirement $SegReq$ (including $s_{\max}$, a predefined upper-bound of the number of segments per block), segments file F into $\mathcal{F} = \{m_{ij}\}$, $i \in [1, l], j \in [1, s], s_i \in [1, s_{\max}]$, i.e., F is segmented into a total of l blocks, with the ith block having s_i segments. In our settings, every file segment should of the same size $\eta \in (0, p)$ and as large as possible (see [15]). Since $|p| = 20$ bytes is used in a BLS signature with 80-bit security (sufficient in practice), $\eta = 20$ bytes is a common choice. According to $s_{\max}$, a set $U = \{u_k \in \mathbb{Z}_p\} k \in [1, s_{\max}]$ is chosen so that the client can compute the HLAs σ_i for each block: $\sigma_i = (H(m_i) \prod_{j=1}^{s_i} u_j^{m_{ij}})^\alpha$ which constitute the ordered set $\Phi = \{\sigma_i\}_{i \in [1,l]}$. This is similar to signing a message with BLS signature. The client also generate a root R based on construction of an RMHT T over $H(m_i)$ and compute $sig = (H(R))^\alpha$. Finally, let $u = (u_1 \| \ldots \| u_{s_{\max}})$, the client compute the file tag for F as $t = name\|n\|u\|\mathrm{Sig}_{ssk}\ (name\|n\|u)$ and then output $\{\mathcal{F}, \Phi, T, R, sig, t\}$.

4.3.3 Prepare for Authorization

The client asks (her choice of) TPA for its ID *VID* (for security, *VID* is used for authorization only). TPA will then return its ID, encrypted with the client's public key. The client will then compute $sig_{AUTH} = \mathrm{Sig}_{ssk}(AUTH\|t\|VID)$ and sends sig_{AUTH} along with the auditing delegation request to TPA for it to compose a challenge later on.

Different from existing schemes, after the execution of the above two algorithms, the client will keep the RMHT 'skeleton' with only ranks of each node and indices of each file block to reduce fine-grained update requests to block-level operations. We will show how this can be done in Section 4.4. The client then sends $\{\mathcal{F}, t, \Phi, sig, AUTH\}$ to CSS and deletes $\{F, \mathcal{F}, t, \Phi, sig\}$ from its local storage. The CSS will construct an RMHT T based on m_i and keep T stored with $\{\mathcal{F}, t, \Phi, sig, AUTH\}$ for later verification, which should be identical to the tree spawned at client-side just a moment ago.

4.3.4 Verifiable Data Updating

Same as *Setup*, this process will also be between client and CSS. We discuss 5 types of block-level updates (operations) that will affect T: $\mathcal{PM}$, $\mathcal{M}$, $\mathcal{D}$, $\mathcal{J}$ and $\mathcal{SP}$ (see *Definition 1*). We will discuss how these requests can form fine-grained update requests in general in Section 4.4.

The verifiable data update process for a $\mathcal{PM}$-typed update is as follows (see Fig. 3):

1. The client composes an update quest $UpdateReq$ defined in Section 4.2 and sends it to CSS.

2. CSS executes the following algorithm:

$PerformUpdate(UpdateReq, \mathcal{F})$: CSS parses $UpdateReq$ and get $\{\mathcal{PM}, i, o, m_{new}\}$. When $Type = \mathcal{PM}$, CSS will update m_i and T accordingly, then output $P_{update} = \{m_i, \Omega_i, R', sig\}$ (note that Ω_i stays the same during the update) and the updated file $\mathcal{F}'$.

Upon finishing of this algorithm, CSS will send P_{update} to the client.

3. After receiving P_{update}, the client executes the following algorithm:

$VerifyUpdate(pk, P_{update})$: The client computes m_i' using $\{m_i, UpdateReq\}$, then parse P_{update} to $\{m_i, \Omega_i, R', sig\}$,

1. There are other possible operations such as block merging $\mathcal{ME}$—two blocks need to be merged into the first block before the second block is deleted, and data moving $\mathcal{MV}$—move a part of data from one block to another, if the size of the second block does not exceed $s_{\max} \cdot \eta$ after this update. However, the fine-grained update requests discussed in this paper do not involve these operations, thus we will omit them in our current discussion. We will leave the problem of how to exploit them in future work.

2. Most exponential operations in this paper are modulo p. Therefore, for simplicity, we will use g^α instead of $g^\alpha \bmod p$ unless otherwise specified.

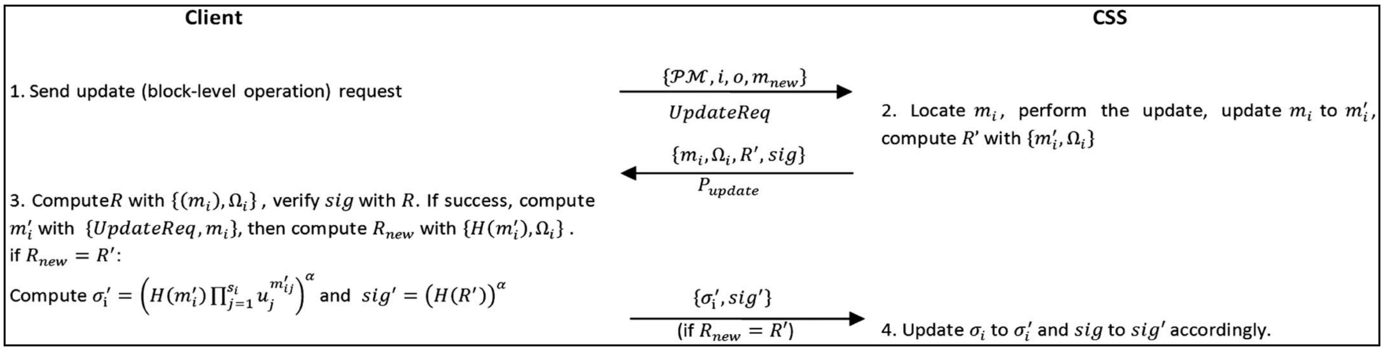
Client
CSS
1. Send update (block-level operation) request
$\{\mathcal{PM}, i, o, m_{new}\}$
$UpdateReq$
2. Locate m_i, perform the update, update m_i to m_i', compute R' with $\{m_i', \Omega_i\}$
$\{m_i, \Omega_i, R', sig\}$
P_{update}
3. Compute$R$ with $\{(m_i), \Omega_i\}$, verify sig with R. If success, compute m_i' with $\{UpdateReq, m_i\}$, then compute R_{new} with $\{H(m_i'), \Omega_i\}$.
if $R_{new} = R'$:
Compute $\sigma_i' = \left(H(m_i')\prod_{j=1}^{s_i} u_j^{m_{ij}'}\right)^\alpha$ and $sig' = (H(R'))^\alpha$
$\{\sigma_i', sig'\}$
(if $R_{new} = R'$)
4. Update σ_i to σ_i' and sig to sig' accordingly.

Fig. 3. Verifiable PM-typed Data Update in our scheme.

compute R (and $H(R)$) and R_{new} use $\{m_i, \Omega_i\}$ and $\{m_i', \Omega_i\}$ respectively. It verifies sig use $H(R)$, and check if $R_{new} = R'$. If either of these two verifications fails, then output $FALSE$ and return to CSS, otherwise output $TRUE$.

If the output of the algorithms is $TRUE$, then the client computes $\sigma_i' = (H(m_i')\prod_{j=1}^{S_1} u_j^{m_{ij}'})^\alpha$ and $sig' = (H(R'))^\alpha$ then sends $\{\sigma_i', sig'\}$ to CSS.

4. The CSS will update σ_i to σ_i' and sig to sig' accordingly and delete $\mathcal{F}$ if it receives $\{\sigma_i', sig'\}$, or it will run $PerformUpdate()$ again if it receives $FALSE$. A cheating CSS will fail the verification and constantly receive $FASLE$ until it performed the update as the client requested.

Due to their similarity to the process described above, other types of operations are only briefly discussed as follows. For whole-block operations $\mathcal{M}$, $\mathcal{D}$, and $\mathcal{J}$, as in model in the existing work [6], the client can directly compute σ_i' without retrieving data from the original file F stored on CSS, thus the client can send σ_i' along with $UpdateReq$ in the first phase. For responding to an update request, CSS only needs to send back $H(m_i)$ instead of m_i. Other operations will be similar to where $Type = \mathcal{PM}$. For an $\mathcal{SP}$-typed update, in addition to updating m_i to m_i', a new block m^* needs to be inserted to T after m_i'. Nonetheless, as the contents in m^* is a part of the old m_i, the CSS still needs to send m_i back to the client. The process afterwards will be just similar to a $\mathcal{PM}$-typed upgrade, with an only exception that the client will compute R_{new} using $\{m_i', h(m^*), \Omega_i\}$ to compare to R', instead of using $\{m_i', \Omega_i\}$ as in the $\mathcal{PM}$-typed update.

4.3.5 *Challenge, Proof Generation and Verification*

In our setting, TPA must show CSS that it is indeed authorized by the file owner before it can challenge a certain file.

1. TPA runs the following algorithm:

$GenChallenge(Acc, pk, sig_{AUTH})$: According to the accuracy required in this auditing, TPA will decide to verify c out of the total l blocks. Then, a challenge message $chal = \{sig_{AUTH}, \{VID\}_{PK_{CSS}}, \{i, v_i\}_{i\in I}\}$ is generated where VID is TPA's ID, I is a randomly selected subset of $[1, l]$ with c elements and $\{v_i \in \mathbb{Z}_p\}_{i\in I}$ are c randomly-chosen coefficients. Note that VID is encrypted with the CSS's public key PK_{CSS} so that CSS can later decrypt $\{VID\}_{PK_{CSS}}$ with the corresponding secret key.

TPA then sends $chal$ to CSS.

2. After receiving $chal$, CSS will run the following algorithm:

$GenProof(pk, F, sig_{AUTH}, \Phi, chal)$: Let $w = \max\{s_i\}_{i\in I}$. CSS will first verify sig_{AUTH} with $AUTH$, t, VID and the client's public key spk, and output $REJECT$ if it fails. Otherwise, CSS will compute $\mu_k = \sum_{i\in I} v_i m_{ik}, k \in [1, w]$ and $\sigma = \prod_{i\in I}\sigma_i^{v_i}$ and compose the proof P as $= \{\{\mu_k\}_{k\in[1,w]}, \{H(m_i), \Omega_i\}_{i\in I}, sig\}$, then ouput P. Note that during the computation of μ_k, we will let $m_{ik} = 0$ if $k > s_i$.

After execution of this algorithm, CSS will send P to TPA.

3. After receiving P, TPA will run the following algorithm:

$Verify(pk, chal, P)$: TPA will compute R using $\{H(m_i), \Omega_i\}$ and then verify sig use public keys g and v by comparing $e(sig, g)$ with $(H(R), v)$. If they are equal, let $\omega = \prod_{i\in I} H(m_i)^{v_i} \cdot \prod_{k\in[1,w]} u_k^{\mu_k}$, TPA will further check if $e(\sigma, g)$ equals $e(\omega, v)$, which is similar to verifying a BLS signature. If all the two equations hold then the algorithm returns $TRUE$, otherwise it returns $FALSE$.

An illustration of Challenge and Verification processes can be found in Fig. 4.

4.4 Analysis on Fine-Grained Dynamic Data Updates

Following the settings in our proposed scheme, we now define a fine-grained update request for an outsourced file divided into l variable-sized blocks, where each block is consisted of $s_i \in [1, s_{\max}]$ segments of a fixed size η each. Assume an RMHT T is built upon $\{m_i\}_{i\in[1,l]}$ for authentication, which means T must keep updated with each RMHT operation for CSS to send back the root R for the client to verify the correctness of this operation (see Section 4.3). We now try to define and categorize all types of fine-grained updates, and then analyze the RMHT operations with $Type = \mathcal{PM}, \mathcal{M}, \mathcal{D}, \mathcal{J}$ or $\mathcal{SP}$ that will be invoked along with the update of the data file.

Definition 2 (Fine-Grained Data Update Request). *A fine-grained update request is defined as $FReq = \{o, len, m_{new}\}$, where o indicates the starting offset of this update in F, len indicates the data length after o that needs to be updated (so that $\{o, len\}$ can characterize an exact proportion of the original file F that needs to be updated, which we will later call m_{old}), and m_{new} is the new message to be inserted into F from offset o.*

We assume the data needed to be obsolete and the new data to be added shares a common starting offset o in F, as

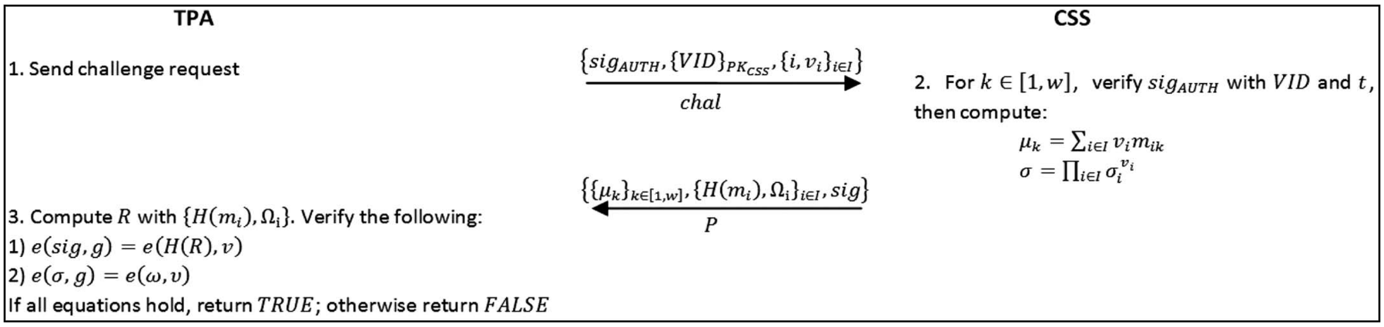
TPA
CSS
1. Send challenge request
$\{sig_{AUTH}, \{VID\}_{PK_{CSS}}, \{i, v_i\}_{i \in I}\}$
$chal$
2. For $k \in [1, w]$, verify sig_{AUTH} with VID and t, then compute:
$\mu_k = \sum_{i \in I} v_i m_{ik}$
$\sigma = \prod_{i \in I} \sigma_i^{v_i}$
$\{\{\mu_k\}_{k \in [1,w]}, \{H(m_i), \Omega_i\}_{i \in I}, sig\}$
P
3. Compute R with $\{H(m_i), \Omega_i\}$. Verify the following:
1) $e(sig, g) = e(H(R), v)$
2) $e(\sigma, g) = e(\omega, v)$
If all equations hold, return $TRUE$; otherwise return $FALSE$

Fig. 4. Challenge, Proof Generation and Verification in our scheme.

otherwise it can be split into multiple updates defined in *Definition 2* commencing in sequence. We now introduce a rule to follow during all update processes:

Condition 1 (Block Size Limits in Updates). *An update operation must not cause the size of any block to exceed $s_{\max}$; After any operation, a block that has 0 bit data remaining must be deleted from T.*

Detailed analysis can be found in Appendix C, which can be summarized as the following theorem:

Theorem 1. *Any valid fine-grained update request that is in the form of $\{o, len, m_{new}\}$ can either directly belong to, or be split into some smaller requests that belong to, the following 5 types of block-level update requests: $\mathcal{PM}$, $\mathcal{M}$, $\mathcal{J}$, $\mathcal{D}$ and $\mathcal{SP}$.*

Proof. See Appendix C. □

Through the analysis above, we know that a large number of small updates, no matter insert, delete or modification, will always invoke a large number of $\mathcal{PM}$ operations. We now try to optimize $\mathcal{PM}$ operations in the next section to make it more efficient.

4.5 Modification for Better Support of Small Updates

Although our proposed scheme can support fine-grained update requests, the client still needs to retrieve the entire file block from CSS to compute the new HLA, in the sense that the client is the only party that has the secret key α to compute the new HLA but clients do not have F stored locally. Therefore, the additional cost in communication will be huge for frequent updates. In this section, we will propose a modification to address this problem, utilizing the fact that CSS only needs to send back data in the block that stayed unchanged.

The framework we use here is identical to the one used in our scheme introduced in Section 4.2 (which we will also name as 'the basic scheme' hereafter). Changes are made in $PerformUpdate$ and $VerifyUpdate$; Setup, Challenge, Proof Generation and Verification phases are same as in our basic scheme. Therefore, we will only describe the two algorithms in the following phase:

4.5.1 Verifiable Data Updating

We also discuss $\mathcal{PM}$ operations here first.

$PerformUpdate$: After CSS has received the update request $UpdateReq$ from the client, it will parse it as $\{\mathcal{PM}, I, o, m_{new}\}$ and use $\{o, |m_{new}|\}$ to gather the sectors that are not involved in this update, which we denote as $\{m_{ij}\}_{j \in M}$. CSS will then perform the update to get m_i', then compute R', then send the proof of update $P_{update} = \{\{m_{ij}\}_{j \in M}, H(m_i), \Omega_i, R', sig\}$ to the client.

$VerifyUpdate$: After the client received $H(m_i)$, it will first compute R using $H(m_i), \Omega_i$ and verify sig, then it will compute m_i' using $\{\{m_{ij}\}_{j \in M}, m_{new}\}$ and then compute R_{new} with $\{m_i', \Omega_i\}$ and compare R_{new} with R'. If $R_{new} = R'$, then the client will return $\{\sigma_i', sig'\}$ to CSS for it to update accordingly.

For an $\mathcal{SP}$ operation the process will be the same to our basic scheme as there are no new data inserted into T, therefore the retrieving of the entire data block is inevitable when computations of σ_i' and σ^* are required. For other types of operations, no old data is involved in new blocks; therefore the processes will also remain the same. The process is shown in Fig. 5.

4.6 Extensions and Generalizations

Our strategy can also be applied in RSA-based PDP or POR schemes to achieve authorized auditing and fine-grained data update requests. As RSA can inherently support variable-sized blocks, the process will be even easier. The batch auditing variation in [6], [7] can also be applied to our scheme, as we did not change the construction of HLAs and the verifications on them.

For the same reason, the random masking strategy for privacy preserving proposed in [7] can also be incorporated into our scheme to prevent TPA from parsing the challenged file blocks through a series of integrity proofs to a same set of blocks. Alternatively, we can also restrict the number of challenges to the same subset of data blocks. When data updates are frequent enough, the success rate of this attack will drop dramatically, because there is a high probability that one or many of the challenged blocks have already updated before c challenges are completed, which is the reason we did not incorporate this strategy into our scheme.

5 Security Analysis

In this section, the soundness and security of our scheme is discussed separately in phases, as the aim and behavior of the malicious adversary in each phase of our scheme is different.

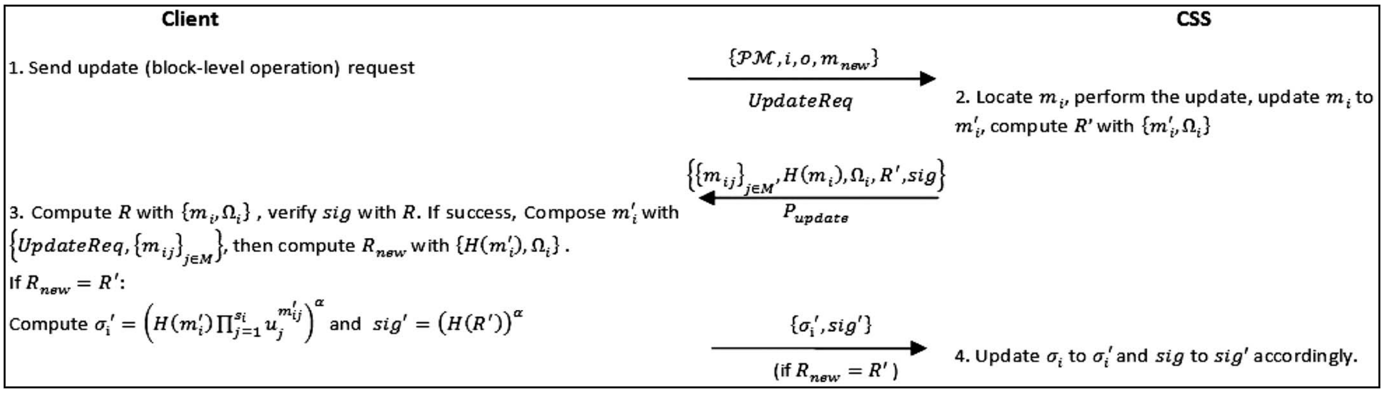
Client
CSS
1. Send update (block-level operation) request
$\{\mathcal{PM}, i, o, m_{new}\}$
$UpdateReq$
2. Locate m_i, perform the update, update m_i to m_i', compute R' with $\{m_i', \Omega_i\}$
$\{\{m_{ij}\}_{j\in M}, H(m_i), \Omega_i, R', sig\}$
P_{update}
3. Compute R with $\{m_i, \Omega_i\}$, verify sig with R. If success, Compose m_i' with $\{UpdateReq, \{m_{ij}\}_{j\in M}\}$, then compute R_{new} with $\{H(m_i'), \Omega_i\}$.
If $R_{new} = R'$:
Compute $\sigma_i' = \left(H(m_i')\prod_{j=1}^{s_i} u_j^{m_{ij}'}\right)^\alpha$ and $sig' = (H(R'))^\alpha$
$\{\sigma_i', sig'\}$
(if $R_{new} = R'$)
4. Update σ_i to σ_i' and sig to sig' accordingly.

Fig. 5. Verifiable PM-typed Data Update in our modified scheme.

5.1 Challenge and Verification

In the challenge/verification process of our scheme, we try to secure the scheme against a malicious CSS who tries to cheat the verifier TPA about the integrity status of the client's data, which is the same as previous work on both PDP and POR. In this step, aside from the new authorization process (which will be discussed in detail later in this section), the only difference compared to [6] is the RMHT and variable-sectored blocks. Therefore, the security of this phase can be proven through a process highly similar with [6], using the same framework, adversarial model and interactive games defined in [6]. A detailed security proof for this phase is therefore omitted here.

5.2 TPA Authorization

Security of the new authorization strategy in our scheme is based on the existential unforgeability of the chosen signature scheme. We first define the behavior of a malicious third-party auditor.

Definition 3 (Malicious TPA). *A malicious TPA is a third party who aims at challenging a user's data stored on CSS for integrity proof without this user's permission. The malicious TPA has access to the entire network.*

According to this definition, none of the previous data auditing schemes is resilient against a malicious TPA. Now, in our scheme, we have the following theorem:

Theorem 2. *Through the authorization process, no malicious TPA can cause the CSS to respond with an integrity proof P over an arbitrary subset of file F, namely $m_i, i \in I$, unless a negligible probability.*

Proof. See Appendix D. □

From this theorem, we can see that the security of a public auditing scheme is strengthened by adding the authorization process. In fact, the scheme is now resilient against malicious or pretended auditing requests, as well as potential DDOS attacks launched by malicious auditors.

For even higher security, the client may mix in a nonce to the authorization message to make every auditing message distinct, so that no one can utilize a previous authorization message. However, this setting may not be appropriate for many scenarios, as the client must stay online when each auditing happens.

5.3 Verifiable Data Updating

In the verifiable updating process, the main adversary is the untrustworthy CSS who did not carry out the data update successfully, but still manages to return a satisfactory response to the client thereafter. We now illustrate the security of this phase of our scheme in the following theorem:

Theorem 3. *In the verifiable update process in both our basic scheme and the modification, CSS cannot provide the client with the satisfactory result, i.e., R' cannot match the R_{new} computed by the client with $\{H(m_i'), \Omega_i\}$, if CSS did not update the data as requested.*

Proof. See Appendix D. □

Note that in the verifiable update process, data retrieval is a part of the verifiable update process. According to *Assumption 1*, CSS will respond this query with the correct m_{ii}. If not with *Assumption 1*, it is recommended to independently retrieve $\{m_{ij}\}_{j\in M}$ before the update so that CSS cannot cheat the client intentionally, as it cannot distinguish whether the following update is based on this retrieval.

If CSS can be trusted even more, the client may let CSS compute $(u_j^{m_{ij}})^\alpha$ (where m_{ij} are the sectors that did not change) and send it back to the client, then the client will be able to compute σ' using it along with m_{new} and $H(m_i')$. This will keep the communication cost of this phase on a constantly low level. However, as the CSS is only considered semi-trusted and it is difficult for the client to verify $(\mu_j^{m_{ij}})^\alpha$ without m_{ij}, this assumption is unfortunately too strong for the majority of scenarios.

6 Evaluation and Experimental Results

We have provided an overall evaluation and comparison in Appendix E.

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 [26] which virtualizes the infrastructure and allows it to provide unified computing and storage resources. Upon virtualized data centers, Hadoop [27] is installed to facilitate the MapReduce programming model and distributed file system. Moreover, we installed OpenStack open source cloud platform [28] which is responsible for global management, resource scheduling, task distribution and interaction with users.

We implemented both our scheme and its modification on U-Cloud, using a virtual machine with 36 CPU cores, 32GB RAM and 1TB storage in total. As in previous work [6], [12], we also used a 1GB randomly generated dataset for testing. The scheme is implemented under 80-bit security, i.e., $\eta = |p| = 160$ bits. As the number of sectors s (per block) is one of the most influential metrics to overall performance, we will use it as our primary metrics. For saving of the first wave of allocated storage, we used $s_i = s_{\max}$ in the initial data splitting and uploading. Note that $s_{\max}$ decides the total number of blocks for an arbitrary $|F|$. However, according to [10], the number of authenticated blocks is a constant with respect to a certain percentage of file tampered and a certain success rate of detection, therefore we will not take the number of audited blocks as our primary variable of measurement. All experimental results are an average of 20 runs.

We first tested how $s_{\max}$ can influence the size of proof P, which is missing in former schemes [6], [7]. From Fig. 6, we can see that generally the proof size decreases when $s_{\max}$ increases, because the average depth of leaf nodes m_i of T decreases when $s_{\max}$ increases to a certain level, especially when right after the initial uploading of $\mathcal{F}$. Note that the storage of HLA and RMHT at CSS side will also decrease with the increase of the average number of blocks. Therefore, a relatively large $s_{\max}$ (but not too large, which we will discuss along with the third experiment) is recommended in our dynamic setting.

Second, we tested the storage overhead for small insertions. Without support for fine-grained updates, every small insertion will cause creation of a whole new block and update of related MHT nodes, which is why our scheme has efficiency advantage. We compared our scheme against a representative (and also recent) public auditing scheme [6]. For comparison, we extended the older scheme a bit to let it support the communication-storage trade-off introduced in [15] so that it can support larger file blocks with multiple (but only a predefined constant number of) sectors each. The updates chosen for experiments are $10 * 140$ Bytes and $10 * 280$ Bytes, filled with random data. Results are shown in Figs. 7 and 8. For updates of the same total size, the increased storage on CSS for our scheme stays constant, while in the extended old scheme [6] (see Section 3.2.2) the storage increases linearly with the increase in size of the affected block. These results demonstrated that our scheme with fine-grained data update support can incur significantly lower storage overhead (down to $0.14\times$ in our test scenarios) for small insertions when compared to existing scheme.

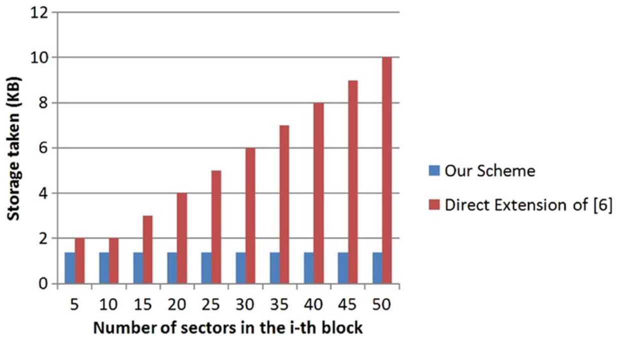

Fig. 7. Comparison of the total storage overhead invoked by 10$*$ 140-byte insertions to the i-th block in our scheme, as opposed to the direct extension of [6].

Third, we investigated the performance improvement of the modification introduced in Section 4.5. We used 3 pieces of random data with sizes of 100 bytes, 140 bytes and 180 bytes, respectively, to update several blocks that contain 10 to 50 standard 20-byte sectors each. Data retrieval is a key factor of communication overheads in the verifiable update phase. For each update, we recorded the total amount of data retrieval for both our modified scheme and our basic scheme. The results in comparison are shown in Fig. 9. We can see that our modified scheme always has better efficiency with respect to data-retrieval-invoked communication overheads, and the advantage is more significant for larger updates. However, for an update of the same size, the advantage will decrease with the increase of $|s_i|$ where a larger number of sectors in the original file are needed to be retrieved. Therefore, the block size needs to be kept low if less communication in verifiable updates is highly demanded.

From the experimental results on small updates, we can see that our scheme can incur significantly lower storage overhead while our modified scheme can dramatically reduce communication overheads compared to the existing scheme. In practice, the important parameter $s_{\max}$ should

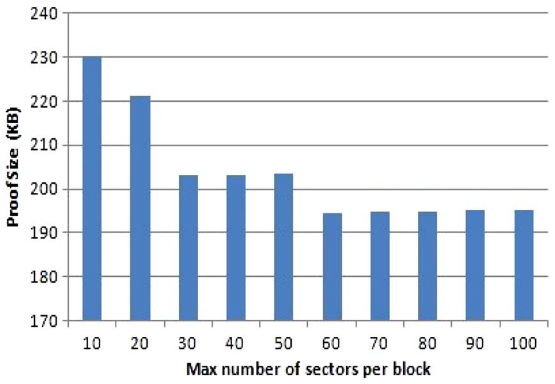

Fig. 6. Communication overhead invoked by an integrity proof with 80-bit security under different $s_{\max}$ for a 1GB data.

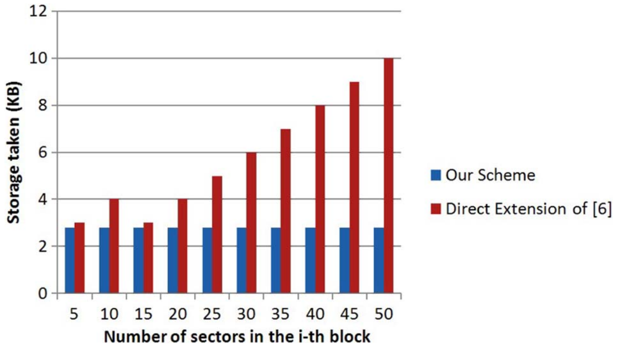

Fig. 8. Comparison of the total storage overhead invoked by 10$*$ 280-byte insertions to the i-th block in our scheme, as opposed to the direct extension of [6].

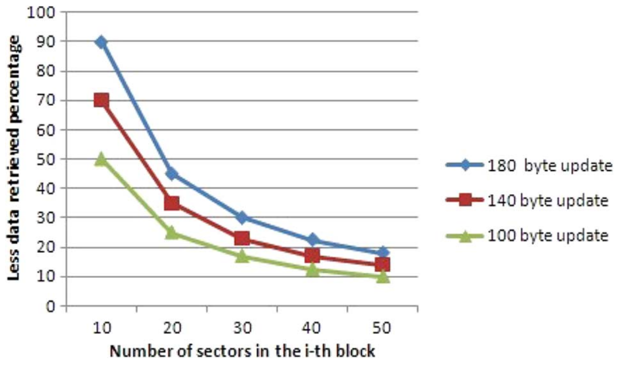

Fig. 9. Percentage in saving of communication overhead in data retrieval in the modified scheme, compared to our basic scheme.

be carefully chosen according to different data size and different efficiency demands in storage or communications. For example, for general applications with a similar scale (1GB per dataset and frequent 140-byte updates), a choice of $s_{\max} = 30$ will allow the scheme to incur significantly lowered overheads in both storage and communications during updates. Additional analysis regarding efficiency can be found in Appendix F.

7 Conclusion and Future Work

In this paper, we have provided a formal analysis on possible types of fine-grained data updates and proposed a scheme that can fully support authorized auditing and fine-grained update requests. Based on our scheme, we have also proposed a modification that can dramatically reduce communication overheads for verifications of small updates. Theoretical analysis and experimental results have demonstrated that our scheme can offer not only enhanced security and flexibility, but also significantly lower overheads for big data applications with a large number of frequent small updates such as applications in social media and business transactions.

Based on the contributions of this paper on improved data auditing, we plan to further investigate the next step on how to improve other server-side protection methods for efficient data security with effective data confidentiality and availability. Besides, we also plan to investigate auditability-aware data scheduling in cloud computing. As data security is also considered as a metric of quality-of-service (QoS) along with other metrics such as storage and computation, a highly efficient security-aware scheduling scheme will play an essential role under most cloud computing contexts.

Acknowledgment

This research work is partly supported by Australian Research Council under Linkage Project LP0990393.

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, ''Cloud Computing and Emerging IT Platforms: Vision, Hype, Reality for Delivering Computing as the 5th Utility,'' *Future Gen. Comput. Syst.*, vol. 25, no. 6, pp. 599-616, June 2009.

[2] 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,'' *Commun. ACM*, vol. 53, no. 4, pp. 50-58, Apr. 2010.

[3] Customer Presentations on Amazon Summit Australia, Sydney, 2012, accessed on: March 25, 2013. [Online]. Available: http://aws.amazon.com/apac/awssummit-au/

[4] J. Yao, S. Chen, S. Nepal, D. Levy, and J. Zic, ''TrustStore: Making Amazon S3 Trustworthy With Services Composition,'' in *Proc. 10th IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2010, pp. 600-605.

[5] D. Zissis and D. Lekkas, ''Addressing Cloud Computing Security Issues,'' *Future Gen. Comput. Syst.*, vol. 28, no. 3, pp. 583-592, Mar. 2011.

[6] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, ''Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 847-859, May 2011.

[7] C. Wang, Q. Wang, K. Ren, and W. Lou, ''Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing,'' in *Proc. 30st IEEE Conf. on Comput. and Commun. (INFOCOM)*, 2010, pp. 1-9.

[8] G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik, ''Scalable and Efficient Provable Data Possession,'' in *Proc. 4th Int'l Conf. Security and Privacy in Commun. Netw. (SecureComm)*, 2008, pp. 1-10.

[9] 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 Trans. Inf. Syst. Security*, vol. 14, no. 1, May 2011, Article 12.

[10] G. Ateniese, R.B. Johns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, ''Provable Data Possession at Untrusted Stores,'' in *Proc. 14th ACM Conf. on Comput. and Commun. Security (CCS)*, 2007, pp. 598-609.

[11] R. Curtmola, O. Khan, R.C. Burns, and G. Ateniese, ''MR-PDP: Multiple-Replica Provable Data Possession,'' in *Proc. 28th IEEE Conf. on Distrib. Comput. Syst. (ICDCS)*, 2008, pp. 411-420.

[12] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, ''Dynamic Provable Data Possession,'' in *Proc. 16th ACM Conf. on Comput. and Commun. Security (CCS)*, 2009, pp. 213-222.

[13] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, ''Cooperative Provable Data Possession for Integrity Verification in Multi-Cloud Storage,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 12, pp. 2231-2244, Dec. 2012.

[14] A. Juels and B.S. Kaliski Jr., ''PORs: Proofs of Retrievability for Large Files,'' in *Proc. 14th ACM Conf. on Comput. and Commun. Security (CCS)*, 2007, pp. 584-597.

[15] H. Shacham and B. Waters, ''Compact Proofs of Retrievability,'' in *Proc. 14th Int'l Conf. on Theory and Appl. of Cryptol. and Inf. Security (ASIACRYPT)*, 2008, pp. 90-107.

[16] S. Nepal, S. Chen, J. Yao, and D. Thilakanathan, ''DIaaS: Data Integrity as a Service in the Cloud,'' in *Proc. 4th Int'l Conf. on Cloud Computing (IEEE CLOUD)*, 2011, pp. 308-315.

[17] Y. He, S. Barman, and J.F. Naughton, ''Preventing Equivalence Attacks in Updated, Anonymized Data,'' in *Proc. 27th IEEE Int'l Conf. on Data Engineering (ICDE)*, 2011, pp. 529-540.

[18] E. Naone, ''What Twitter Learns From All Those Tweets,'' in *Technology Review*, Sept. 2010, accessed on: March 25, 2013. [Online]. Available: http://www.technologyreview.com/view/420968/what-twitter-learns-from-all-those-tweets/

[19] X. Zhang, L.T. Yang, C. Liu, and J. Chen, ''A Scalable Two-Phase Top-Down Specialization Approach for Data Anonymization Using MapReduce on Cloud,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 363-373, Feb. 2014.

[20] S.E. Schmidt, ''Security and Privacy in the AWS Cloud,'' presented at the Presentation Amazon Summit Australia, Sydney, Australia, May 2012, accessed on: March 25, 2013. [Online]. Available: http://aws.amazon.com/apac/awssummit-au/

[21] 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 Gen. Comput. Syst.*, vol. 29, no. 5, pp. 1300-1308, July 2013.

[22] 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 Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1192-1202, June 2013.