International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol. 3, Issue. 6, Nov - Dec. 2013 pp-3786-3791 ISSN: 2249-6645

Implement Public Audit ability and Data Dynamics with Security in Cloud Computing

[1]P. Srinivasa Rao, [2]K. Prasada Rao
[1]*(M. Tech), Dept. of CSE, AITAM (Tekkali), Srikakulam, A.P, India*
[2]*Sr. asst. Prof., Dept. of CSE, AITAM (Tekkali), Srikakulam, A.P, India*

Abstract: *Cloud Computing has been visualized as the next-generation architecture of IT Enterprise. It moves the application software and databases to the centralized large data centers, where the management of the data and services may not be fully trustworthy. This unique paradigm brings about many new security challenges, which have not been well understood. This work studies the problem of ensuring the integrity of data storage in Cloud Computing. In particular, we consider the task of allowing a third party auditor (TPA), on behalf of the cloud client, to verify the integrity of the dynamic data stored in the cloud. The introduction of TPA eliminates the involvement of the client through the auditing of whether his data stored in the cloud are naturally* ***intact****, which can be important in achieving economies of scale for Cloud Computing. The support for data dynamics via the most general forms of data operation, such as block modification, insertion, and deletion, is also a significant step toward practicality, since services in Cloud Computing are not limited to archive or backup data only. While prior works on ensuring remote data integrity often lacks the support of either* ***public Auditability*** *or dynamic data operations, this paper achieves both. We first identify the difficulties and potential security problems of direct extensions with fully dynamic data updates from prior works and then show how to construct an dignified verification scheme for the* ***seamless -integration*** *of these two salient features in our protocol design. In particular, to achieve efficient data dynamics, we improve the existing proof of storage models by manipulating the* ***classic Merkle Hash Tree*** *construction for block tag authentication. To support efficient handling of multiple auditing tasks, we further explore the technique of* ***bilinear aggregate signature*** *to extend our main result into a multi-user setting, where TPA can perform multiple auditing tasks simultaneously. Extensive security and performance analysis show that the proposed schemes are highly efficient and provably secure.*

Index terms: *seamless-integration, intact, classic merkle hash tree, bilinear aggregate*

I. INTRODUCTION

SEVERAL trends are opening up the era of Cloud Computing, which is an Internet-based development and use of computer technology. The ever cheaper and more powerful processors, together with the "software as a service" (SaaS) computing architecture, are transforming data centers into pools of computing service on a huge scale. Meanwhile, the increasing network bandwidth and reliable yet flexible network connections make it even possible that clients can now subscribe high-quality.

Services from data and software that reside solely on remote data centers. Although envisioned as a promising service platform for the Internet, this new data storage paradigm in "Cloud" brings about many challenging design issues which have profound influence on the security and performance of the overall system. One of the biggest concerns with cloud data storage is that of data integrity verification at un trusted servers. For example, the storage service provider, which experiences Byzantine failures occasionally, may decide to hide the data errors from the clients for the benefit of their own. What is more serious is that for saving money and storage space the service provider might neglect to keep or deliberately delete rarely accessed data files which belong to an ordinary client. Consider the large size of the outsourced electronic data and the client's constrained resource capability, the core of the problem can be generalized as how can the client find an efficient way to perform periodical integrity verifications without the local copy of data files.

In order to solve the problem of data integrity checking, many schemes are proposed under different systems and security models. In all these works, great efforts are made to design solutions that meet various requirements:high scheme efficiency, stateless verification, unbounded use of queries and retrievability of data, etc. Considering the role of the verifier in the model, all the schemes presented before fall into two categories: private auditability and public auditability. Although schemes with private audit ability can achieve higher scheme efficiency, public audit ability allows anyone, not just the client (data owner), to challenge the cloud server for correctness of data storage while keeping no private information. Then, clients are able to delegate the evaluation of the service performance to an independent third party auditor (TPA), without devotion of their computation resources. In the cloud, the clients themselves are unreliable or may not be able to afford the overhead of performing frequent integrity checks. Moreover, for efficiency consideration, the outsourced data themselves should not be required by the verifier for the verification purpose. Another major concern among previous designs is that of supporting dynamic data operation for cloud data storage applications. In Cloud Computing, the remotely stored electronic data might not only be accessed but also updated by the clients, e.g., through block modification, deletion, insertion, etc. Unfortunately, the state of the art in the context of remote data storage mainly focus on static data files and the importance of this dynamic data updates has received limited attention so far . Moreover, as will be shown later, the direct extension of the current provable data possession (PDP) or proof of retrievability (PoR) schemes to support data dynamics may lead to security loopholes. Although there are many difficulties faced by researchers, it is well believed that supporting dynamic data operation can be of vital importance to the practical application of storage outsourcing services. In view of the key role of public auditability and data dynamics for cloud data storage, we propose an efficient construction for the seamless integration of these two components in the protocol design. Our contribution can be summarized as follows: 1.We motivate the public auditing system of data storage security in Cloud Computing, and propose a protocol supporting for fully dynamic data operations, especially to support block insertion, which is missing in most existing schemes. 2. We extend our scheme to support scalable and efficient public auditing in Cloud Computing. In particular, our scheme achieves batch auditing where multiple delegated auditing tasks from different users can be performed simultaneously by the TPA. 3. We prove the security of our proposed construction and justify the performance of our scheme through concrete implementation and comparisons with the state of the art.

1.1 Existing System:

The perspective of data security, which has always been an important aspect in quality of service, Cloud Computing inevitably poses new challenging security threats for number of reasons. Firstly, traditional cryptographic primitives for the purpose of data security protection cannot be directly adopted due to the users' loss control of data under Cloud Computing. Therefore, verification of correct data storage in the cloud must be conducted without explicit knowledge of the whole data. Considering various kinds of data for each user stored in the cloud and the demand of long term continuous assurance of their data safety, the problem of verifying correctness of data storage in the cloud becomes even more challenging simply called as a integrity of data Secondly, Cloud Computing is not just a third party data warehouse. The data stored in the cloud may be frequently updated by the users, including data dynamics like insertion, deletion, modification, appending, reordering, etc. To ensure storage correctness under dynamic data update is hence of paramount importance.

II. PROBLEM STATEMENTS

2.1 System Model

A representative network architecture for cloud data storage is illustrated in Fig. 1. Three different network entities can be identified as follows:

Client: an entity, which has large data files to be stored in the cloud and relies on the cloud for data maintenance and computation, can be either individual consumers or organizations;

. **Cloud Storage Server (CSS)**: an entity, which is managed by Cloud Service Provider (CSP), has significant storage space and computation resource to maintain the clients' data;

. **Third Party Auditor(TPA):** an entity, which has expertise and capabilities that clients do not have, is trusted to assess and expose risk of cloud storage services on behalf of the clients upon request. In the cloud paradigm, by putting the large data files on the remote servers, the clients can be relieved of the burden of storage and computation. As clients no longer possess their data locally, it is of critical importance for the clients to ensure that their data are being correctly stored and maintained. That is, clients should be equipped with certain security means so that they can periodically verify the correctness of the remote data even without the existence of local copies. In case that clients do not necessarily have the time, feasibility or resources to monitor their data, they can delegate the monitoring task to a trusted TPA. In this paper, we only consider verification schemes with public auditability: any TPA in possession of the public key can act as a verifier. We assume that TPA is unbiased while the server is untrusted. For application purposes, the clients may interact with the cloud servers via CSP to access or retrieve their prestored data. More importantly, in practical scenarios, the client may frequently perform block-level operations on the data files. The most general forms of these operations we consider in this paper are modification, insertion, and deletion. Note that we don't address the issue of data privacy in this paper, as the topic of data privacy in Cloud Computing is orthogonal to the problem we study here.

2.2 Design Goals:

Our design goals can be summarized as the following:

1. Public verification for storage perfectness assurance
2. Real time operations on information like modify, insert, update, delete etc…

2.3 Proposed System:

In this paper, we propose an effective and flexible distributed scheme with explicit dynamic data support to find the correctness of users' data in the cloud. We rely on erasure correcting code in the file distribution preparation to provide redundancies and guarantee the data dependability. This construction drastically reduces the communication and storage overhead as compared to the traditional replication-based file distribution techniques. By utilizing the homomorphic token with distributed verification of erasure-coded data, our scheme achieves the storage correctness insurance as well as data error localization: whenever data corruption has been detected during the storage correctness verification, our scheme can almost guarantee the simultaneous localization of data errors, i.e., the identification of the misbehaving server(s).

Bilinear map: A bilinear map is a map e : G _ G ! GT, where G is a Gap Diffie-Hellman (GDH) group and GT is another multiplicative cyclic group of prime order p with the following properties :

1) Computable: there exists an efficiently computable algorithm for computing e;

2) Bilinear: for all h1; h2 2 G and a; b 2 ZZp; eðha1; hb2Þ ¼ eðh1; h2Þab;

3) Non degenerate: eðg gÞ 6¼ 1, where g is a generator of G.

Merkle hash tree: A Merkle Hash Tree (MHT) is a well studied authentication structure [17], which is intended to efficiently and securely prove that a set of elements are undamaged and unaltered. It is constructed as a binary tree where the leaves in the MHT are the hashes of authentic data values. Fig. 2 depicts an example of authentication. The verifier with the authentic hr requests for fx2; x7g and requires the authentication of the received blocks. The prover provides the verifier with the auxiliary authentication information (AAI) _2 ¼< hðx1Þ; hd > and _7 ¼ <hðx8Þ; he>. The verifier can then verify x2 and x7 by first computing hðx2Þ; hðx7Þ; hc ¼ hðhðx1Þkhðx2ÞÞÞ; hf ¼ hðhðx7Þkhðx8ÞÞÞ; ha ¼ hðhckhdÞ; hb ¼ hðhekhf Þ a n d hr ¼hðhakhbÞ, and then checking if the calculated hr is the same as the authentic one. MHT is commonly used to authenticate the values of data blocks. However, in this paper, we further employ MHT to authenticate both the values and the positions of data blocks. We treat the leaf nodes as the left-to-right sequence, so any leaf node can be uniquely determined by following this sequence and the way of computing the root in MHT.

III. MODELING THE SYSTEM

In this section, we present our security protocols for cloud data storage service with the aforementioned research goals in mind. We start with some basic solutions aiming to provide integrity assurance of the cloud data and discuss their demerits. Then, we present our protocol which supports public auditability and data dynamics. We also show how to extent our main scheme to support batch auditing for TPA upon delegations from multiusers.

3.1 Definition

$(\mathrm{pk,sk}) \leftarrow \mathrm{KeyGen}(1^k)$.. This probabilistic algorithm is run by the client. It takes as input security parameter 1^k, and returns public key pk and private key sk. $(\Phi, \mathrm{sig}_{sk}(\mathrm{H(R)})) \leftarrow \mathrm{SigGen(sk;\ F)}$ This algorithm is run by the client. It takes as input private key sk and a file F which is an ordered collection of blocks $\{m_i\}$ and outputs the signature set Φ, which is an ordered collection of signatures $\{\sigma_i\} on \{m_i\}$ It also outputs metadata—the signature $\mathrm{sig}_{sk}(\mathrm{H(R)})$ of the root R of a Merkle hash tree. In our construction, the leaf nodes of the Merkle hash tree are hashes of $H(m_i)$ straightforwardly as the verification covers all the data blocks. However, the number of verifications allowed to be performed in this solution is limited by the number of secret keys. Once the keys are exhausted, the data owner has to retrieve the entire file of F from the server in order to compute new MACs, which is usually impractical due to the huge communication overhead. Moreover, public auditability is not supported as the private keys are required for verification. Another basic solution is to use signatures instead of MACs to obtain public auditability. The data owner precomputes the signature of each block $m^i\,(i \in [1, n])$ and sends both F and the signatures to the cloud server for storage. To verify the correctness of F, the data owner can adopt a spot-checking approach, i.e., requesting a number of randomly selected blocks and their corresponding signatures to be returned. This basic solution can provide probabilistic assurance of the data correctness and support public auditability. However, it also severely suffers from the fact that a considerable number of original data blocks should be retrieved to ensure a reasonable detection probability, which again could result in a large communication overhead and greatly affects system efficiency. Notice that the above solutions can only support the case of static data, and none of them can deal with dynamic data updates.

3.2 Construction

To effectively support public auditability without having to retrieve the data blocks themselves, we resort to the homomorphic authenticator technique. Homomorphic authenticators are unforgeable metadata generated from individual data blocks, which can be securely aggregated in such a way to assure a verifier that a linear combination of data blocks is correctly computed by verifying only the aggregated authenticator. In our design, we propose to use PKC-based homomorphic authenticator (e.g., BLS signature or RSA signature-based authenticator) to equip the verification protocol with public auditability. In the following description, we present the BLS-based scheme to illustrate our design with data dynamics support. As will be shown, the schemes designed under BLS construction can also be implemented in RSA construction. In the discussion of Section 3.4, we show that direct extensions of previous work have security problems, and we believe that protocol design for supporting dynamic data operation is a major challenging task for cloud storage systems. Now we start to present the main idea behind our scheme. We assume that file F (potentially encoded using Reed-Solomon codes) is divided into n blocks $m_1, m_2, ..., m_n$ where $m^i \in z_p$ and p is a large prime. Let $e : G * G \rightarrow G_T$ be a bilinear map, with a hash function $H : \{0,1\}* \rightarrow G$, viewed as a random oracle. Let g be the generator of G. h is a cryptographic hash function. The procedure of our protocol execution is as follows:

3.3 Setup

The client's public key and private key are generated by Invoking key gen (.). By running sig gen (.), the data file F is preprocessed, and the homomorphic authenticators together with metadata are produced. $keyGen(1^k)$ The client generates a random signing key pair (spk,ssk). Choose a random $\alpha \rightarrow Z_p$ and compute $v \leftarrow g^{\alpha}$ .The secret key is $sk = (\alpha, ssk)$ and the public key is pk =(v,spk) SigGen(sk,f);FÞ. Given $F = (m_1, m_2, ..., m_n)$, the client chooses a random element $u \longleftarrow G$. Let $t = name \| n \| u \| ssig_{ssk}(name \| n \| u)$ be the file tag for F. Then, the client computes signature σ_i for each block $m_i (i = 1,2,...,n)$ as $\sigma_i \leftarrow (H(m_i).u^{mi})^{\alpha}$ Denote the set of signatures by $\Phi = \{\sigma_i\}, 1 \le i \le n$. The client then generates a root R based on the construction of the MHT, where the leave nodes of the tree are an ordered set of hashes of "file tags". $H(m_i)(i = 1,2....,n)$ Next, the client signs the root Runder the private key

$\alpha : sig_{sk}(H(R)) \leftarrow (H(R))^{\alpha}$. The client sends $\{F, t, \phi, sig_{sk}(H(R))\}$ to the server and deletes $\{F, t, \phi, sig_{sk}(H(R))$ from its local storage.

3.4 Default Integrity Verification

The client or TPA can verify the integrity of the outsourced data by challenging the server. Before challenging, the TPA first use spk to verify the signature on t. If the verification fails, reject by emitting FALSE; otherwise, recover u. To generate the message “chal,” the TPA (verifier) picks a random c-element subset $I = \{s_1, s_2, ..., s_c\}$ of set [1, n], where we assume $s_1 \leq ... \leq sc$. For each $i \in I$ the TPA chooses a random element $v_i \leftarrow B \subseteq Zp$. The message “chal” specifies the positions of the blocks to be checked in this challenge phase. The verifier sends the to $chal = \{(i, v_i)\}_{s1 \leq i \leq sc}$ the prover (server). Gen proof (f, ϕ ,chal)Upon receiving the challenge $chal = \{(i, v_i)\}_{s1 \leq i \leq sc}$, the server computes

$$\mu = \sum_{i=s}^{s_c} v_i m_i \in z_p and \sigma = \pi \sum_{i=s_1}^{s_2} \sigma_i^{v_i} \in G$$

where both the data blocks and the corresponding signature blocks are aggregated into a single block, respectively. In addition, the prover will also provide the verifier with a small amount of auxiliary information $\{\Omega_i\} s_1 \leq i \leq s_c$, which are the node siblings on the path from the leaves $\{h(H(m_i))\}_{s1 \leq i \leq s_c}$ to the root R of the MHT. The prover responds the verifier with proof $p = \{\mu, \sigma, \{H(m_i), \Omega_i\}_{s1 \leq i \leq s_c,}, sig_{sk}(H(R))\}$ Verify Proof $(pk, chal, P)$. Upon receiving the responses from the pr over, the verifier generates root R using $\{H(m_i), \Omega_i\}_{s1 \leq i \leq sc}$ and authenticates it by checking $e(sig_{sk}(H(R)), g) = e(H(R), g^{\alpha})$ If the authentication fails, the verifier rejects by emitting FALSE. Otherwise, the verifier checks

$$e(\sigma, g). \overset{?}{=} e(\Pi_{i=s_1}^{s_c} H(m_i)^{v_i} .u^{\mu}, v)$$

If so, output TRUE; otherwise FALSE. The protocol is illustrated in Table 1.

PROTOCOLS FOR DEFAULT INTEGRITY VERIFICATION

1. Generate a random set $\{(i, v_i)\}_{i \in I}$
2. Compute $\mu = \sum_i v_i m_i$
3. Compute $\sigma = \prod_i \sigma_i^{v_i}$
4. Compute R using $\{H(m_i)_i \Omega_i\}_{i \in I}$
5. Verify $sig_{sk}(H(R))$ and output False if fail
6. Verify $\{m_i\}_{i \in I}$

PROTOCOL FOR PROVABLE DATA UPDATE

1. Generate $\sigma_i^1 = (H(m_i^1)\mu^{m_i^1})^{\alpha}$
2.Update F and compute R^1
3.Compute R using $\{H(m_i), \Omega_i\}$
4 .**Verify** $sig_{sk}(H(R))$ and output False if fail
5.compute R_{new} using $\{\Omega_i, H(m_i^1)\}$ verify update by checking $R_{new} = R^1$. sign R^1 if succeed
6.Update $R^1 s$ signature

3.5 Dynamic Data Operation with Integrity Assurance

Now we show how our scheme can explicitly and efficiently handle fully dynamic data operations including data modification (M), data insertion (I), and data deletion (D) for cloud data storage. Note that in the following descriptions, we assume that the file F and the signature ϕ have already been generated and properly stored at server. The root metadata R has been signed by the client and stored at the cloud server, so that anyone who has the client’s public key can challenge the correctness of data storage. Data Modification: We start from data modification, which is one of the most frequently used operations in cloud data storage. A basic data modification operation refers to the replacement of specified blocks with new ones. Suppose the client wants to modify the Ith block m_i to $m_i^{|}$. The protocol procedures are described in Table 2. At start, based on the new $m_i^{|}$, the client generates the corresponding signature $\sigma_i^{|} = (H(m_i^{|}).u^{m_i^{|}})^{\alpha}$. Then, he constructs an update request message "$update = (m, i, m_i^{|}, \alpha_i^{|})$".and sends to the server, where M denotes the modification operation. Upon

receiving the request, the server runs $ExecUpdate(F,\Phi,update)$. Specifically, the server 1) replaces the block m_i with $m_i^|$ and outputs $F^|$; 2) replaces the α_i with $\sigma_i^|$ and outputs ϕ; and 3) replaces $H(m_i)$ with $H(m_i^|)$ in the Merkle hash tree construction and generates the new root R' (see the example in Fig. 3). Finally, the
server responses the client with a proof for this operation,
P_{update}=$(\Omega_i, H(m_i), sig_{sk}(H(R), R^1)$, where Ω_i is the AAI for authentication of m_i . After receiving the proof for modification operation from server, the client first generates root R using $\{\Omega_i, H(m_i)\}$ and authenticates the AAI or R by checking $e(sig_{sk}(H(R)), g) \overset{?}{=} e(H(R), g^{\alpha})$. If it is not true, output FALSE, otherwise the client can now check whether the server has performed the modification as required or not, by further computing the new root value using $\{\Omega_i, H(m_i^1)\}$ and comparing it with R0. If it is not true output FALSE, otherwise output TRUE. Then, the client signs the new root metadata R^1 by $sig_{sk}(H(R^1))$ and sends it to the server for update. Finally, the client executes them default integrity verification protocol. If the output is TRUE, delete $sig_{sk}(H(R^1))$; update and m_i^1 from its local storage.

Data Insertion: Compared to data modification, which does not change the logic structure of client's data file,
another general form of data operation, data insertion, refers to inserting new blocks after some specified positions in the data file F.Suppose the client wants to insert block after the i^{th} block m_i. The protocol procedures are similar to the data modification case (see Table 2, now m_i^1 can be seen as m^*).At start, based on m_ the client generates the corresponding signature $\sigma^* = (H(m^*), u^{m^*})^{\alpha}$. Then, he constructs an update request message "$update = (I, i, m^*, \sigma^*)$" and sends to the server, where I denotes the insertion operation. Upon receiving the request, the server runs $ExecUpdate(F,\Phi,update)$. Specifically, the server 1) stores m^* and adds a leaf $h(H(m^*))$ "after" leaf $h(H(m_i))$ in the Merkle hash tree and outputs F^1; 2) adds the σ^* into the signature set and outputs Φ^1 ; and 3) generates the new root R^1 based on the updated Merkle hash tree. Finally, the server responses the client with a proof for this operation, P_{update}=$(\Omega_i, H(m_i), sig_{sk}(H(R), R^1)$, where Ω_i is the AAI for authentication of m_i in the old tree. An example of block insertion, to insert $h(H(m^*))$ after leaf node $h(H(m_2))$, only node $h(H(m^*))$ and an internal node C is added to the original tree, where $h_c = h(h(H(m_2)) \| h(H(m^*)))$
. After receiving the proof for insert operation from server, the client first generates root R using
$\{\Omega_i, H(m_i)\}$ and then authenticates the AAI or R by checking if $e(sig_{sk}(H(R)), g) \overset{?}{=} e(H(R), g^{\alpha})$. If it is not true, output FALSE, otherwise the client can now check whether the server has performed the insertion as required or not, by further computing the new root value using $\{\Omega_i, H(m_i), H(m^*)\}$ and comparing it with R^1. If it is not true, output FALSE, otherwise output TRUE. Then, the client signs the new root metadata $R^1$by $sig_{sk}(H(R^1))$and sends it to the server for update. Finally, the client executes the default integrity verification protocol. If the output is TRUE, delete $sig_{sk}(H(R^1))$, P_{update} and m^* from its local storage.

Data Deletion: Data deletion is just the opposite operation of data insertion. For single block deletion, it refers to deleting the specified block and moving all the latter blocks one block forward. Suppose the server receives
the update request for deleting block m_i, it will delete mi from its storage space, delete the leaf node $h(H(m_i))$ in the MHT and generate the new root metadata R0 (see the example in Fig. 5). The details of the protocol procedures
are similar to that of data modification and insertion, which are thus omitted here.

IV. SECURITY ANALYSIS

In this section, we evaluate the security of the proposed scheme under the security model defined in Section 2.2. we consider a file F after Reed-Solomon coding.

Definition 1 (CDH Problem). The Computational Diff ie-Hellman problem is that, given g, g^x $g^y \in g$ for unknown x,y$\in z_p$ to compute g ^{xy}.We say that the (t, ρ)-CDH assumption holds in G if no t time algorithm has the non-negligible probability $\in$ in solving the CDH problem. A proof-of-retrievability protocol is sound if any cheating prover that convinces the verification algorithm that it is storing a file F is actually storing that file, which we define in saying that it yields up the file F to an extractor algorithm which interacts with it using the proof-of-retrievability protocol. We say that the adversary (cheating server) is $\in$admissible if it convincingly answers an $\in$fraction of verification challenges. We formalize the notion of an extractor and then give a precise definition for soundness.

Theorem 1. Suppose a cheating prover on an n-block file F is well-behaved in the sense above, and that it is ρ admissible. Let w=1/#B+ $(\rho n)^{l}$/(n-c+1)^{e} Then, provided that ρ -w is positive and non-negligible, it is possible to recover a _-fraction of the encoded file blocks in O (n/(e- ρ)) interactions with cheating prover and in O (n^{2} + (1+en^{2})(n)/(e-w)) time overall.

Proof. The verification of the proof-of-retrievability is similar to [4], we omit the details of the proof here. The difference in our work is to replace H (i) with H (m_{i}) such that secure update can still be realized without including the index information. These two types of tags are used for the same purpose (i.e., to prevent potential attacks), so this change will not affect the extraction algorithm defined in the proof-of-retrievability. We can also prove that extraction always succeeds against a well-behaved cheating prover, with the same probability analysis given in .

Theorem 2. Given a fraction of the n blocks of an encoded file F, it is possible to recover the entire original file F with all but negligible probability.

Proof. Based on the rate ρ Reed-Solomon codes, this result can be easily derived, since any ρ fraction of encoded file blocks suffices for decoding. The security proof for the multi client batch auditing is similar to the single-client case, thus omitted here. variant of this relationship. P=1- ρ^{e} Under this setting, we quantify the extra cost introduced by the support of dynamic data in our scheme into server computation, verifier computation as well as communication overhead.

V. CONCLUSION

To ensure cloud data storage security, it is critical to enable a TPA to evaluate the service quality from an objective and independent perspective. Public auditability also allows clients to delegate the integrity verification tasks to TPA while they themselves can be unreliable or not be able to commit necessary computation resources performing continuous verifications. Another major concern is how to construct verification protocols that can accommodate dynamic data files. In this paper, we explored the problem of providing simultaneous public auditability and data dynamics for remote data integrity check in Cloud Computing. Our construction is deliberately designed to meet these two important goals while efficiency being kept closely in mind. To achieve efficient data dynamics, we improve the existing proof of storage models by manipulating the classic Merkle Hash Tree construction for block tag authentication. To support efficient handling of multiple auditing tasks, we further explore the technique of bilinear aggregate signature to extend our main result into a multiuser setting, where TPA can perform multiple auditing tasks simultaneously. Extensive security and performance analysis show that the proposed scheme is highly efficient and provably secure.

REFERENCES

[1]. Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing," Proc. 14th European Symp. Research in Computer Security (ESORICS '09), pp. 355-370, 2009.
[2]. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," Proc. 14th ACM Conf. Computer and Comm. Security (CCS 07), pp. 598-609, 2007.
[3]. A. Juels and B.S. Kaliski Jr., "Pors: Proofs of Retrievability for Large Files," Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), pp. 584-597, 2007.
[4]. H. Shacham and B. Waters, "Compact Proofs of Retrievability," Proc. 14th Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '08), pp. 90-107, 2008.
[5]. K.D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," Report 2008/175, Cryptology ePrint Archive, 2008.
[6]. M. Naor and G.N. Rothblum, "The Complexity of Online Memory Checking," Proc. 46th Ann. IEEE Symp. Foundations of Computer Science (FOCS '05), pp. 573-584, 2005.
[7]. E.-C. Chang and J. Xu, "Remote Integrity Check with Dishonest Storage Server," Proc. 13th European Symp. Research in Computer Security (ESORICS '08), pp. 223-237, 2008.
[8]. M.A. Shah, R. Swaminathan, and M. Baker, "Privacy-Preserving Audit and Extraction of Digital Contents," Report 2008/186,Cryptology ePrint Archive, 2008.
[9]. A. Oprea, M.K. Reiter, and K. Yang, "Space-Efficient Block Storage Integrity," Proc. 12th Ann. Network and Distributed System Security Symp. (NDSS '05), 2005.
[10]. T. Schwarz and E.L. Miller, "Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage," Proc. 26th IEEE Int'l Conf. Distributed Computing Systems (ICDCS '06), p. 12, 2006.
[11]. Q. Wang, K. Ren, W. Lou, and Y. Zhang, "Dependable and Secure Sensor Data Storage with Dynamic Integrity Assurance," Proc. IEEE INFOCOM, pp. 954-962, Apr. 2009.
[12]. G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," Proc. Fourth Int'l Conf. Security and Privacy in Comm. Networks (SecureComm '08), pp. 1-10, 2008.
[13]. C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring Data Storage Security in Cloud Computing," Proc. 17th Int'l Workshop Quality of Service (IWQoS '09), 2009.
[14]. C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," Proc. 16th ACM Conf. Computer and Comm. Security (CCS '09), 2009.
[15]. K.D. Bowers, A. Juels, and A. Oprea, "Hail: A High-Availability and Integrity Layer for Cloud Storage," Proc. 16th ACM Conf. Computer and Comm. Security (CCS '09), pp. 187-198, 2009.
[16]. D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," Proc. Seventh Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '01), pp. 514-532, 2001.
[17]. R.C. Merkle, "Protocols for Public Key Cryptosystems," Proc. IEEE Symp. Security and Privacy, pp. 122-133, 1980.
[18]. S. Lin and D.J. Costello, Error Control Coding, second ed., Prentice- Hall, 2004.