Provable Multicopy Dynamic Data Possession in Cloud Computing Systems

Ayad F. Barsoum and M. Anwar Hasan

***Abstract*— Increasingly more and more organizations are opting for outsourcing data to remote cloud service providers (CSPs). Customers can rent the CSPs storage infrastructure to store and retrieve almost unlimited amount of data by paying fees metered in gigabyte/month. For an increased level of scalability, availability, and durability, some customers may want their data to be replicated on multiple servers across multiple data centers. The more copies the CSP is asked to store, the more fees the customers are charged. Therefore, customers need to have a strong guarantee that the CSP is storing all data copies that are agreed upon in the service contract, and all these copies are consistent with the most recent modifications issued by the customers. In this paper, we propose a map-based provable multicopy dynamic data possession (MB-PMDDP) scheme that has the following features: 1) it provides an evidence to the customers that the CSP is not cheating by storing fewer copies; 2) it supports outsourcing of dynamic data, i.e., it supports block-level operations, such as block modification, insertion, deletion, and append; and 3) it allows authorized users to seamlessly access the file copies stored by the CSP. We give a comparative analysis of the proposed MB-PMDDP scheme with a reference model obtained by extending existing provable possession of dynamic single-copy schemes. The theoretical analysis is validated through experimental results on a commercial cloud platform. In addition, we show the security against colluding servers, and discuss how to identify corrupted copies by slightly modifying the proposed scheme.**

***Index Terms*— Cloud computing, data replication, outsourcing data storage, dynamic environment.**

I. Introduction

OUTSOURCING data to a remote cloud service provider (CSP) allows organizations to store more data on the CSP than on private computer systems. Such outsourcing of data storage enables organizations to concentrate on innovations and relieves the burden of constant server updates and other computing issues. Moreover, many authorized users can access the remotely stored data from different geographic locations making it more convenient for them.

Once the data has been outsourced to a remote CSP which may not be trustworthy, the data owners lose the direct control over their sensitive data. This lack of control raises new formidable and challenging tasks related to data confidentiality and integrity protection in cloud computing. The confidentiality issue can be handled by encrypting sensitive data before outsourcing to remote servers. As such, it is a crucial demand of customers to have a strong evidence that the cloud servers still possess their data and it is not being tampered with or partially deleted over time. Consequently, many researchers have focused on the problem of provable data possession (PDP) and proposed different schemes to audit the data stored on remote servers.

PDP is a technique for validating data integrity over remote servers. In a typical PDP model, the data owner generates some metadata/information for a data file to be used later for verification purposes through a *challenge-response* protocol with the remote/cloud server. The owner sends the file to be stored on a remote server which may be untrusted, and deletes the local copy of the file. As a proof that the server is still possessing the data file in its original form, it needs to correctly compute a response to a challenge vector sent from a verifier — who can be the original data owner or a trusted entity that shares some information with the owner. Researchers have proposed different variations of PDP schemes under different cryptographic assumptions; for example, see [1]–[9].

One of the core design principles of outsourcing data is to provide dynamic behavior of data for various applications. This means that the remotely stored data can be not only accessed by the authorized users, but also updated and scaled (through block level operations) by the data owner. PDP schemes presented in [1]–[9] focus on only *static* or warehoused data, where the outsourced data is kept unchanged over remote servers. Examples of PDP constructions that deal with *dynamic* data are [10]–[14]. The latter are however for a *single* copy of the data file. Although PDP schemes have been presented for *multiple* copies of *static* data, see [15]–[17], to the best of our knowledge, this work is the first PDP scheme directly dealing with *multiple* copies of *dynamic* data. In Appendix A, we provide a summary of related work.

When verifying multiple data copies, the overall system integrity check fails if there is one or more corrupted copies. To address this issue and recognize which copies have been

Manuscript received October 18, 2013; revised May 22, 2014 and December 11, 2014; accepted December 11, 2014. Date of publication December 18, 2014; date of current version January 22, 2015. This work was supported by the Natural Sciences and Engineering Research Council of Canada. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. C.-C. Jay Kuo.

A. F. Barsoum is with the Department of Computer Science, St. Mary's University at Texas, San Antonio, TX 78228 USA (e-mail: afekry@engmail.uwaterloo.ca).

M. A. Hasan is with the Department of Electrical and Computer Engineering, University of Waterloo, ON N2L 3G1, Canada (e-mail: ahasan@ece.uwaterloo.ca).

This paper has supplementary downloadable material at http://ieeexplore.ieee.org, provided by the authors. The file consists of Appendices A, B, C, and D. The material is 1 MB in size.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2014.2384391

corrupted, we discuss a slight modification to be applied to the proposed scheme.

A. Main Contributions

Our contributions can be summarized as follows:

- We propose a map-based provable multi-copy dynamic data possession (MB-PMDDP) scheme. This scheme provides an adequate guarantee that the CSP stores all copies that are agreed upon in the service contract. Moreover, the scheme supports outsourcing of dynamic data, *i.e.*, it supports block-level operations such as block modification, insertion, deletion, and append. The authorized users, who have the right to access the owner's file, can seamlessly access the copies received from the CSP.
- We give a thorough comparison of MB-PMDDP with a reference scheme, which one can obtain by extending existing PDP models for dynamic single-copy data. We also report our implementation and experiments using Amazon cloud platform.
- We show the security of our scheme against colluding servers, and discuss a slight modification of the proposed scheme to identify corrupted copies.

Remark 1: Proof of retrievability (POR) is a complementary approach to PDP, and is stronger than PDP in the sense that the verifier can reconstruct the entire file from responses that are reliably transmitted from the server. This is due to encoding of the data file, for example using *erasure codes*, before outsourcing to remote servers. Various POR schemes can be found in the literature, for example [18]–[23], which focus on *static* data.

In this work, we do not encode the data to be outsourced for the following reasons. First, we are dealing with dynamic data, and hence if the data file is encoded before outsourcing, modifying a portion of the file requires re-encoding the data file which may not be acceptable in practical applications due to high computation overhead. Second, we are considering economically-motivated CSPs that may attempt to use less storage than required by the service contract through deletion of a few copies of the file. The CSPs have almost no financial benefit by deleting only a small portion of a copy of the file. Third, and more importantly, unlike erasure codes, duplicating data files across multiple servers achieves scalability which is a fundamental customer requirement in CC systems. A file that is duplicated and stored strategically on multiple servers – located at various geographic locations – can help reduce access time and communication cost for users. Besides, a server's copy can be reconstructed even from a complete damage using duplicated copies on other servers.

B. Paper Organization

The remainder of the paper is organized as follow. Our system and assumptions are presented in Section II. The proposed scheme is elaborated in Section III. The performance analysis is shown in Section IV. Section V presents the implementation and experimental results using Amazon cloud platform. How to identify the corrupted copies is discussed in Section VI. Concluding remarks are given in Section VII.

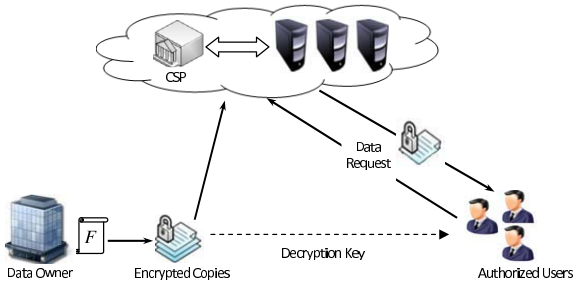

Fig. 1. Cloud computing data storage system model.

II. Our System and Assumptions

A. System Components

The cloud computing storage model considered in this work consists of three main components as illustrated in Fig. 1: (i) a data owner that can be an organization originally possessing sensitive data to be stored in the cloud; (ii) a CSP who manages cloud servers (CSs) and provides paid storage space on its infrastructure to store the owner's files; and (iii) authorized users — a set of owner's clients who have the right to access the remote data.

The storage model used in this work can be adopted by many practical applications. For example, e-Health applications can be envisioned by this model where the patients' database that contains large and sensitive information can be stored on the cloud servers. In these types of applications, the e-Health organization can be considered as the data owner, and the physicians as the authorized users who have the right to access the patients' medical history. Many other practical applications like financial, scientific, and educational applications can be viewed in similar settings.

B. Outsourcing, Updating, and Accessing

The data owner has a file F consisting of m blocks and the CSP offers to store n copies $\{\widetilde{F}_1, \widetilde{F}_2, \ldots, \widetilde{F}_n\}$ of the owner's file on different servers — to prevent simultaneous failure of all copies — in exchange of pre-specified fees metered in GB/month. The number of copies depends on the nature of data; more copies are needed for critical data that cannot easily be reproduced, and to achieve a higher level of scalability. This critical data should be replicated on multiple servers across multiple data centers. On the other hand, non-critical, reproducible data are stored at reduced levels of redundancy. The CSP pricing model is related to the number of data copies.

For data confidentiality, the owner encrypts his data before outsourcing to CSP. After outsourcing all n copies of the file, the owner may interact with the CSP to perform block-level operations on all copies. These operations includes modify, insert, append, and delete specific blocks of the outsourced data copies.

An authorized user of the outsourced data sends a data-access request to the CSP and receives a file copy in an encrypted form that can be decrypted using a secret key shared with the owner. According to the load balancing mechanism used by the CSP to organize the work of the servers, the

data-access request is directed to the server with the lowest congestion, and thus the user is not aware of which copy has been received.

We assume that the interaction between the owner and the authorized users to authenticate their identities and share the secret key has already been completed, and it is not considered in this work.

C. Threat Model

The integrity of customers' data in the cloud may be at risk due to the following reasons. First, the CSP — whose goal is likely to make a profit and maintain a reputation — has an incentive to hide data loss (due to hardware failure, management errors, various attacks) or reclaim storage by discarding data that has not been or is rarely accessed. Second, a dishonest CSP may store fewer copies than what has been agreed upon in the service contact with the data owner, and try to convince the owner that all copies are correctly stored intact. Third, to save the computational resources, the CSP may totally ignore the data-update requests issued by the owner, or not execute them on all copies leading to inconsistency between the file copies. The goal of the proposed scheme is to detect (with *high probability*) the CSP misbehavior by validating the number and integrity of file copies.

D. Underlying Algorithms

The proposed scheme consists of seven polynomial time algorithms: KeyGen, CopyGen, TagGen, PrepareUpdate, ExecUpdate, Prove, and Verify. The data owner runs the algorithms KeyGen, CopyGen, TagGen, and PrepareUpdate. The CSP runs the algorithms ExecUpdate and Prove, while a verifier runs the Verify algorithm.

- $(pk, sk) \leftarrow \mathsf{KeyGen}()$. This algorithm is run by the data owner to generate a public key pk and a private key sk. The private key sk is kept secret by the owner, while pk is publicly known.
- $\widetilde{\mathbb{F}} \leftarrow \mathsf{CopyGen}(CN_i, F)_{1\le i\le n}$. This algorithm is run by the data owner. It takes as input a copy number CN_i and a file F, and generates n copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}_{1\le i\le n}$. The owner sends the copies $\widetilde{\mathbb{F}}$ to the CSP to be stored on cloud servers.
- $\Phi \leftarrow \mathsf{TagGen}(sk, \widetilde{\mathbb{F}})$. This algorithm is run by the data owner. It takes as input the private key sk and the file copies $\widetilde{\mathbb{F}}$, and outputs tags/authenticators set Φ, which is an ordered collection of tags for the data blocks. The owner sends Φ to the CSP to be stored along with the copies $\widetilde{\mathbb{F}}$.
- $(\mathcal{D}', \mathit{UpdateReq}) \leftarrow \mathsf{PrepareUpdate}(\mathcal{D}, \mathit{UpdateInfo})$. This algorithm is run by the data owner to update the outsourced file copies stored by the remote CSP. The input parameters are a previous metadata $\mathcal{D}$ stored on the owner side, and some information *UpdateInfo* about the dynamic operation to be performed on a specific block. The outputs of this algorithm are a modified metadata $\mathcal{D}'$ and an update request *UpdateReq*. This request may contain a modified version of a previously stored block, a new block to be inserted, or a delete command to delete a specific block from the file copies. *UpdateReq* also contains updated (or new) tags for modified (or inserted/appended) blocks, and it is sent from the data owner to the CSP in order to perform the requested update.
- $(\widetilde{\mathbb{F}}', \Phi') \leftarrow \mathsf{ExecUpdate}(\widetilde{\mathbb{F}}, \Phi, \mathit{UpdateReq})$. This algorithm is run by the CSP, where the input parameters are the file copies $\widetilde{\mathbb{F}}$, the tags set Φ, and the request *UpdateReq*. It outputs an updated version of the file copies $\widetilde{\mathbb{F}}'$ along with an updated tags set Φ'. The latter does not require the private key to be generated; just replacement/insertion/deletion of one item of Φ by a new item sent from the owner.
- $\mathbb{P} \leftarrow \mathsf{Prove}(\widetilde{\mathbb{F}}, \Phi, \mathit{chal})$. This algorithm is run by the CSP. It takes as input the file copies $\widetilde{\mathbb{F}}$, the tags set Φ, and a challenge *chal* (sent from a verifier). It returns a proof $\mathbb{P}$ which guarantees that the CSP is actually storing n copies and all these copies are intact, updated, and consistent.
- $\{1, 0\} \leftarrow \mathsf{Verify}(pk, \mathbb{P}, \mathcal{D})$. This algorithm is run by a verifier (original owner or any other trusted auditor). It takes as input the public key pk, the proof $\mathbb{P}$ returned from the CSP, and the most recent metadata $\mathcal{D}$. The output is 1 if the integrity of all file copies is correctly verified or 0 otherwise.

E. Security Requirements

The security of the proposed scheme can be stated using a "game" that captures the data possession property [1], [12], [18]. The data possession game between an adversary $\mathcal{A}$ (acts as a malicious CSP) and a challenger $\mathcal{C}$ (plays the roles of a data owner and a verifier) consists of the following:

- SETUP. $\mathcal{C}$ runs the KeyGen algorithm to generate a key pair (pk, sk), and sends pk to $\mathcal{A}$.
- INTERACT. $\mathcal{A}$ interacts with $\mathcal{C}$ to get the file copies and the verification tags set Φ. $\mathcal{A}$ *adaptively* selects a file F and sends it to $\mathcal{C}$. $\mathcal{C}$ divides the file into m blocks, runs the two algorithms CopyGen and TagGen to create n distinct copies $\widetilde{\mathbb{F}}$ along with the tags set Φ, and returns both $\widetilde{\mathbb{F}}$ and Φ to $\mathcal{A}$.

  Moreover, $\mathcal{A}$ can interact with $\mathcal{C}$ to perform dynamic operations on $\widetilde{\mathbb{F}}$. $\mathcal{A}$ specifies a block to be updated, inserted, or deleted, and sends the block to $\mathcal{C}$. $\mathcal{C}$ runs the PrepareUpdate algorithm, sends the *UpdateReq* to $\mathcal{A}$, and updates the local metadata $\mathcal{D}$. $\mathcal{A}$ can further request challenges $\{chal_i\}_{1\le i\le L}$ for some parameter $L \ge 1$ of $\mathcal{A}$'s choice, and return proofs $\{\mathbb{P}_i\}_{1\le i\le L}$ to $\mathcal{C}$. $\mathcal{C}$ runs the Verify algorithm and provides the verification results to $\mathcal{A}$. The INTERACT step can be repeated *polynomially-many* times.
- CHALLENGE. $\mathcal{A}$ decides on a file F previously used during the INTERACT step, requests a challenge *chal* from $\mathcal{C}$, and generates a proof $\mathbb{P} \leftarrow \mathsf{Prove}(\widetilde{\mathbb{F}}', \Phi, \mathit{chal})$, where $\widetilde{\mathbb{F}}'$ is $\widetilde{\mathbb{F}}$ except that at least one of its file copies (or a portion of it) is missing or tampered with. Upon receiving the proof $\mathbb{P}$, $\mathcal{C}$ runs the Verify algorithm and if $\mathsf{Verify}(pk, \mathbb{P}, \mathcal{D})$ returns 1, then $\mathcal{A}$ has won the game.

Note that $\mathcal{D}$ is the latest metadata held by $\mathcal{C}$ corresponding to the file F. The CHALLENGE step can be repeated *polynomially-many* times for the purpose of data extraction.

The proposed scheme is secure if the probability that any *probabilistic polynomial-time* (PPT) adversary $\mathcal{A}$ wins the game is negligible. In other words, if a PPT adversary $\mathcal{A}$ can win the game with non-negligible probability, then there exists a *polynomial* time extractor that can repeatedly execute the CHALLENGE step until it extracts the blocks of data copies.

III. Proposed MB-PMDDP Scheme

A. Overview and Rationale

Generating unique differentiable copies of the data file is the core to design a provable multi-copy data possession scheme. Identical copies enable the CSP to simply deceive the owner by storing only one copy and pretending that it stores multiple copies. Using a simple yet *efficient* way, the proposed scheme generates distinct copies utilizing the *diffusion* property of any secure encryption scheme. The diffusion property ensures that the output bits of the ciphertext depend on the input bits of the plaintext in a very complex way, *i.e.*, there will be an unpredictable complete change in the ciphertext, if there is a single bit change in the plaintext [24]. The interaction between the authorized users and the CSP is considered through this methodology of generating distinct copies, where the former can decrypt/access a file copy received from the CSP. In the proposed scheme, the authorized users need only to keep a single secret key (shared with the data owner) to decrypt the file copy, and it is not necessarily to recognize the index of the received copy.

In this work, we propose a MB-PMDDP scheme allowing the data owner to update and scale the blocks of file copies outsourced to cloud servers which may be untrusted. Validating such copies of dynamic data requires the knowledge of the block versions to ensure that the data blocks in all copies are consistent with the most recent modifications issued by the owner. Moreover, the verifier should be aware of the block indices to guarantee that the CSP has inserted or added the new blocks at the requested positions in all copies. To this end, the proposed scheme is based on using a small data structure (metadata), which we call a map-version table.

B. Map-Version Table

The map-version table (MVT) is a small *dynamic* data structure stored on the verifier side to validate the integrity and consistency of all file copies outsourced to the CSP. The MVT consists of three columns: serial number ($\mathcal{SN}$), block number ($\mathcal{BN}$), and block version ($\mathcal{BV}$). The $\mathcal{SN}$ is an indexing to the file blocks. It indicates the *physical* position of a block in a data file. The $\mathcal{BN}$ is a counter used to make a *logical* numbering/indexing to the file blocks. Thus, the relation between $\mathcal{BN}$ and $\mathcal{SN}$ can be viewed as a mapping between the logical number $\mathcal{BN}$ and the physical position $\mathcal{SN}$. The $\mathcal{BV}$ indicates the current version of file blocks. When a data file is initially created the $\mathcal{BV}$ of each block is 1. If a specific block is being updated, its $\mathcal{BV}$ is incremented by 1.

Remark 2: It is important to note that the verifier keeps only <u>*one*</u> table for unlimited number of file copies, *i.e.*, the storage requirement on the verifier side does not depend on the number of file copies on cloud servers. For n copies of a data file of size $|F|$, the storage requirement on the CSP side is $O(n|F|)$, while the verifier's overhead is $O(m)$ for all file copies (m is the number of file blocks).

Remark 3: The MVT is implemented as a linked list to simplify the insertion deletion of table entries. For actual implementation, the $\mathcal{SN}$ is not needed to be stored in the table; $\mathcal{SN}$ is considered to be the entry/table index, *i.e.*, each table entry contains just two integers $\mathcal{BN}$ and $\mathcal{BV}$ (8 bytes). Thus, the total table size is $8m$ bytes for all file copies. We further note that although the table size is linear to the file size, in practice the former would be smaller by *several orders of magnitude*. For example, outsourcing unlimited number of file copies of a 1GB-file with 16KB block size requires a verifier to keep MVT of only 512KB ($< 0.05\%$ of the file size). More details on the MVT and how it works will be explained later.

C. Notations

- F is a data file to be outsourced, and is composed of a sequence of m blocks, *i.e.*, $F = \{b_1, b_2, \ldots, b_m\}$.
- $\pi_{key}(\cdot)$ is a pseudo-random permutation (PRP): $key \times \{0,1\}^{\log_2(m)} \rightarrow \{0,1\}^{\log_2(m)}$.[1]
- $\psi_{key}(\cdot)$ is a pseudo-random function (PRF): $key \times \{0,1\}^* \rightarrow \mathbb{Z}_p$ (p is a large prime).
- *Bilinear Map/Pairing:* Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the properties [25]:
  1) *Bilinear:* $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$ $\forall$ $u \in \mathbb{G}_1, v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$
  2) *Non-Degenerate:* $\hat{e}(g_1, g_2) \neq 1$
  3) *Computable:* there exists an efficient algorithm for computing $\hat{e}$
- $\mathcal{H}(\cdot)$ is a map-to-point hash function : $\{0,1\}^* \rightarrow \mathbb{G}_1$.
- E_K is an encryption algorithm with strong *diffusion* property, *e.g.*, AES.

Remark 4: Homomorphic linear authenticators (HLAs) [18], [22], [26] are basic building blocks in the proposed scheme. Informally, the HLA is a fingerprint/tag computed by the owner for each block b_j that enables a verifier to validate the data possession on remote servers by sending a challenge vector $C = \{c_1, c_2, \ldots, c_r\}$. As a response, the servers can homomorphically construct a tag authenticating the value $\sum_{j=1}^{r} c_j \cdot b_j$. The response is validated by a verifier, and accepted only if the servers honestly compute the response using the owner's file blocks. The proposed scheme utilizes the BLS HLAs [18].

D. MB-PMDDP Procedural Steps

- *Key Generation:* Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a bilinear map and g a generator of $\mathbb{G}_2$. The data owner runs the

[1]The number of file blocks (m) will be changed due to dynamic operations on the file. We use HMAC-SHA-1 with 160-bit output to allow up to 2^{160} blocks in the file.

KeyGen algorithm to generate a private key $x \in \mathbb{Z}_p$ and a public key $y = g^x \in \mathbb{G}_2$.

- *Generation of Distinct Copies:* For a file $F = \{b_j\}_{1 \le j \le m}$, the owner runs the CopyGen algorithm to create n differentiable copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}_{1 \le i \le n}$, where a copy $\widetilde{F}_i = \{\tilde{b}_{ij}\}_{1 \le j \le m}$. The block $\tilde{b}_{ij}$ is generated by concatenating a copy number i with the block b_j, then encrypting using an encryption scheme E_K, *i.e.*, $\tilde{b}_{ij} = E_K(i||b_j)$. The encrypted block $\tilde{b}_{ij}$ is fragmented into s sectors $\{\tilde{b}_{ij1}, \tilde{b}_{ij2}, \ldots, \tilde{b}_{ijs}\}$, *i.e.*, the copy $\widetilde{F}_i = \{\tilde{b}_{ijk}\}_{\substack{1 \le j \le m, \\ 1 \le k \le s}}$ where each sector $\tilde{b}_{ijk} \in \mathbb{Z}_p$ for some large prime p. The number of block sectors s depends on the block size and the prime p, where $s = \lceil |\text{block size}|/|p| \rceil$ ($|.|$ is the bit length).

  The authorized users need only to keep a single secret key K. Later, when an authorized user receives a file copy from the CSP, he decrypts the copy blocks, removes the copy index from the blocks header, and then recombines the decrypted blocks to reconstruct the plain form of the received file copy.
- *Generation of Tags:* Given the distinct file copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}$, where $\widetilde{F}_i = \{\tilde{b}_{ijk}\}$, the data owner chooses s random elements $(u_1, u_2, \ldots, u_s) \in_R \mathbb{G}_1$ and runs the TagGen algorithm to generate a tag σ_{ij} for each block $\tilde{b}_{ij}$ as $\sigma_{ij} = (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x \in \mathbb{G}_1$ ($i : 1 \to n$, $j : 1 \to m$, $k : 1 \to s$). In the tag computation, $\mathcal{BN}_j$ is the *logical* number of the block at *physical* position j, $\mathcal{BV}_j$ is the current version of that block, and $ID_F = Filename||n||u_1||\ldots||u_s$ is a unique fingerprint for each file F comprising the file name, the number of copies for this file, and the random values $\{u_k\}_{1 \le k \le s}$. Note that if the data owner decides to use different block size (or different p) for his different files, the number of block sectors s and hence $\{u_k\}_{1 \le k \le s}$ will change. We assume that ID_F is signed with some owner's signing secret key (different than x), and the CSP verifies this signature during different scheme operations to validate the owner's identity.

  In order to reduce storage overhead on cloud servers and lower communication cost, the data owner generates an aggregated tag σ_j for the blocks at the same indices in each copy $\widetilde{F}_i$ as $\sigma_j = \prod_{i=1}^{n} \sigma_{ij} \in \mathbb{G}_1$. Hence, instead of storing mn tags, the proposed scheme requires the CSP to store only m tags for the copies $\widetilde{\mathbb{F}}$. Let us denote the set of aggregated tags as $\Phi = \{\sigma_j\}_{1 \le j \le m}$. The data owner sends $\{\widetilde{\mathbb{F}}, \Phi, ID_F\}$ to the CSP, and deletes the copies and the tags from its local storage. The MVT is stored on the local storage of the owner (or any trusted verifier).
- *Dynamic Operations on the Data Copies:* The dynamic operations are performed at the block level via a request in the general form $\langle ID_F, BlockOp, j, \{b_i^*\}_{1 \le i \le n}, \sigma_j^* \rangle$, where ID_F is the file identifier and *BlockOp* corresponds to block modification (denoted by BM), block insertion (BI), or block deletion (BD). The parameter j indicates the index of the block to be updated, $\{b_i^*\}_{1 \le i \le n}$ are the new block values for all copies, and σ_j^* is the new aggregated tag for the new blocks.
  - *Modification:* For a file $F = \{b_1, b_2, \ldots, b_m\}$, suppose the owner wants to modify a block b_j with b'_j for all file copies $\widetilde{\mathbb{F}}$. The owner runs the PrepareUpdate algorithm to do the following:
    1) Updates $\mathcal{BV}_j = \mathcal{BV}_j + 1$ in the MVT
    2) Creates n distinct blocks $\{\tilde{b}'_{ij}\}_{1 \le i \le n}$, where $\tilde{b}'_{ij} = E_K(i||b'_j)$ is fragmented into s sectors $\{\tilde{b}'_{ij1}, \tilde{b}'_{ij2}, \ldots, \tilde{b}'_{ijs}\}$
    3) Creates a new tag σ'_{ij} for each block $\tilde{b}'_{ij}$ as $\sigma'_{ij} = (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}'_{ijk}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\sigma'_j = \prod_{i=1}^{n} \sigma'_{ij} \in \mathbb{G}_1$
    4) Sends a modify request $\langle ID_F, \mathsf{BM}, j, \{\tilde{b}'_{ij}\}_{1 \le i \le n}, \sigma'_j \rangle$ to the CSP

    Upon receiving the request, the CSP runs the ExecUpdate algorithm to do the following:
    1) Replaces the block $\tilde{b}_{ij}$ with $\tilde{b}'_{ij} \forall i$, and constructs updated file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1 \le i \le n}$
    2) Replaces σ_j with σ'_j in the set Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2, \ldots, \sigma'_j, \ldots, \sigma_m\}$.
  - *Insertion:* Suppose the owner wants to insert a new block $\hat{b}$ *after* position j in a file $F = \{b_1, b_2, \ldots, b_m\}$, *i.e.*, the newly constructed file is $F' = \{b_1, b_2, \ldots, b_j, \hat{b}, \ldots, b_{m+1}\}$, where $b_{j+1} = \hat{b}$. In the proposed MB-PMDDP scheme, the *physical* block index $\mathcal{SN}$ is not included in the block tag. Thus, the insertion operation can be performed without recomputing the tags of all blocks that have been shifted after inserting the new block. Embedding the physical index in the tag results in unacceptable computation overhead, especially for large data files. To perform the insertion of a new block $\hat{b}$ after position j in all file copies $\widetilde{\mathbb{F}}$, the owner runs the PrepareUpdate algorithm to do the following:
    1) Constructs a new table entry $\langle \mathcal{SN}, \mathcal{BN}, \mathcal{BV} \rangle = \langle j{+}1, (Max\{\mathcal{BN}_j\}_{1 \le j \le m}){+}1, 1 \rangle$, and inserts this entry in the MVT after position j
    2) Creates n distinct blocks $\{\hat{b}_i\}_{1 \le i \le n}$, where $\hat{b}_i = E_K(i||\hat{b})$ is fragmented into s sectors $\{\hat{b}_{i1}, \hat{b}_{i2}, \ldots, \hat{b}_{is}\}$
    3) Creates a new tag $\hat{\sigma}_i$ for each block $\hat{b}_i$ as $\hat{\sigma}_i = (\mathcal{H}(ID_F||\mathcal{BN}_{j+1}||\mathcal{BV}_{j+1}) \cdot \prod_{k=1}^{s} u_k^{\hat{b}_{ik}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\hat{\sigma} = \prod_{i=1}^{n} \hat{\sigma}_i \in \mathbb{G}_1$. Note that $\mathcal{BN}_{j+1}$ is the logical number of the new block with current version $\mathcal{BV}_{j+1} = 1$
    4) Sends an insert request $\langle ID_F, \mathsf{BI}, j, \{\hat{b}_i\}_{1 \le i \le n}, \hat{\sigma} \rangle$ to the CSP

    Upon receiving the insert request, the CSP runs the ExecUpdate algorithm to do the following:
    1) Inserts the block $\hat{b}_i$ after position j in the file copy $\widetilde{F}_i$ $\forall i$, and constructs a new version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1 \le i \le n}$

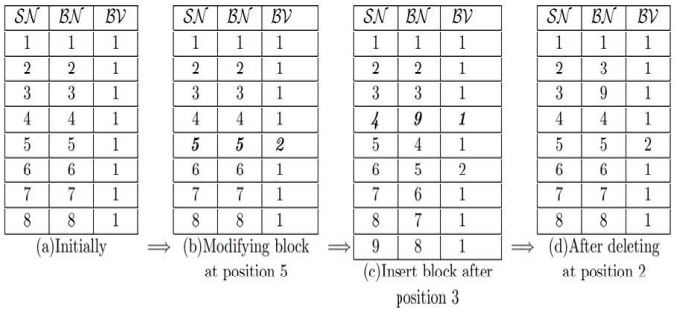

Fig. 2. Changes in the MVT due to different dynamic operations on copies of a file $F = \{b_j\}_{1 \le j \le 8}$.

2) Inserts $\hat{\sigma}$ after index j in Φ, and outputs $\Phi' = \{\sigma_1, \ldots, \sigma_j, \hat{\sigma}, \ldots, \sigma_{m+1}\}$, *i.e.*, $\sigma_{j+1} = \hat{\sigma}$

Remark 5: To prevent the CSP from cheating and using less storage, the modified or inserted blocks for the outsourced copies cannot be identical. To this end, the proposed scheme leaves the control of creating such distinct blocks in the owner hand. This illustrates the linear relation between the work done by the owner during dynamic operations and the number of copies. The proposed scheme assumes that the CSP stores the outsourced copies on *different* servers to avoid simultaneous failure and achieve a higher level of availability. Therefore, even if the CSP is honest to perform part of the owner work, this is unlikely to significantly reduce the communication overhead since the distinct blocks are sent to different servers for updating the copies. The experimental results show that the computation overhead on the owner side due to dynamic block operations is practical.

- *Append:* Block append operation means adding a new block at the end of the outsourced data. It can simply be implemented via insert operation after the last block of the data file.
- *Deletion:* When one block is deleted all subsequent blocks are moved one step forward. To delete a specific data block at position j from all copies, the owner deletes the entry at position j from the MVT and sends a delete request $\langle ID_F, \mathsf{BD}, j, null, null \rangle$ to the CSP. Upon receiving this request, the CSP runs the ExecUpdate algorithm to do the following:
  1) Deletes the blocks $\{\tilde{b}_{ij}\}_{1 \le i \le n}$, and outputs a new version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}_i'\}_{1 \le i \le n}$
  2) Deletes σ_j from Φ and outputs $\Phi' = \{\sigma_1, \sigma_2, \ldots, \sigma_{j-1}, \sigma_{j+1}, \ldots, \sigma_{m-1}.\}$

Fig. 2 shows the changes in the MVT due to dynamic operations on the copies $\widetilde{\mathbb{F}}$ of a file $F = \{b_j\}_{1 \le j \le 8}$. When the copies are initially created (Fig. 2a), $\mathcal{SN}_j = \mathcal{BN}_j$ and $\mathcal{BV}_j = 1$: $1 \le j \le 8$. Fig. 2b shows that $\mathcal{BV}_5$ is incremented by 1 for updating the block at position 5 for all copies. To insert a new block after position 3 in $\widetilde{\mathbb{F}}$, Fig. 2c shows that a new entry $\langle 4, 9, 1 \rangle$ is inserted in the MVT after $\mathcal{SN}_3$, where 4 is the physical position of the newly inserted block, 9 is the new logical block number computed by incrementing the maximum of all previous logical block numbers, and 1 is the version of the new block. Deleting a block at position 2 from all copies requires deleting the table entry at $\mathcal{SN}_2$ and shifting all subsequent entries one position up (Fig. 2d). Note that during all dynamic operations, the $\mathcal{SN}$ indicates the actual physical positions of the data blocks in the file copies $\widetilde{\mathbb{F}}$.

- *Challenge:* For challenging the CSP and validating the integrity and consistency of all copies, the verifier sends c (# of blocks to be challenged) and two fresh keys at each challenge: a PRP(π) key k_1 and a PRF(ψ) key k_2. *Both* the verifier and the CSP use π keyed with k_1 and the ψ keyed with k_2 to generate a set $Q = \{(j, r_j)\}$ of c pairs of random indices and random values, where $\{j\} = \pi_{k_1}(l)_{1 \le l \le c}$ and $\{r_j\} = \psi_{k_2}(l)_{1 \le l \le c}$. The set of random indices $\{j\}$ is the physical positions (serial numbers $\mathcal{SN}$) of the blocks to be challenged.
- *Response:* The CSP runs the Prove algorithm to generate a set $Q = \{(j, r_j)\}$ of random indices and values, and provide an evidence that the CSP is still correctly possessing the n copies in an updated and consistent state. The CSP responds with a proof $\mathbb{P} = \{\sigma, \mu\}$, where $\sigma = \prod_{(j,r_j) \in Q} \sigma_j^{r_j} \in \mathbb{G}_1$, $\mu_{ik} = \sum_{(j,r_j) \in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p$, and $\mu = \{\mu_{ik}\}_{\substack{1 \le i \le n \\ 1 \le k \le s}}$.
- *Verify Response:* Upon receiving the proof $\mathbb{P} = \{\sigma, \mu\}$ from the CSP, the verifier runs the Verify algorithm to check the following verification equation: $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}$

$$\left(\Big[\prod_{(j,r_j) \in Q} \mathcal{H}(ID_F || \mathcal{BN}_j || \mathcal{BV}_j)^{r_j} \Big]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y \right) \quad (1)$$

The verifier uses the set of random indices $\{j\}$ (generated from π) and the MVT to get the logical block number $\mathcal{BN}_j$ and the block version $\mathcal{BV}_j$ of each block being challenged. If the verification equation passes, the Verify algorithm returns 1, otherwise 0.

One can attempt to slightly modify the MB-PMDDP scheme to reduce the communication overhead by a factor of n via allowing the CSP to compute and send $\mu = \{\hat{\mu}_k\}_{1 \le k \le s}$, where $\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik}$. However, this modification enables the CSP to simply cheat the verifier as follows:

$$\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik} = \sum_{i=1}^{n} \sum_{(j,r_j) \in Q} r_j \cdot \tilde{b}_{ijk} = \sum_{(j,r_j) \in Q} r_j \cdot \sum_{i=1}^{n} \tilde{b}_{ijk}$$

Thus, the CSP can just keep the sectors summation $\sum_{i=1}^{n} \tilde{b}_{ijk}$ not the sectors themselves. Moreover, the CSP can corrupt the sectors and the summation is still valid. Therefore, we require the CSP send $\mu = \{\mu_{ik}\}_{\substack{1 \le i \le n \\ 1 \le k \le s}}$, and the summation $\sum_{i=1}^{n} \mu_{ik}$ is done on the verifier side. The challenge response protocol in the MB-PMDDP scheme is summarized in Fig. 3.

Remark 6: The proposed MB-PMDDP scheme supports public verifiability where anyone, who knows the owner's public key but is not necessarily the data owner, can send

Verifier **CSP**

1. Generates a PRP key k_1 and a PRF key k_2.
2. Determines c (# of blocks in the challenge vector)
3. Generates a set $Q = \{(j, r_j)\}$,
   $\{j\} = \pi_{k_1}(l)_{1 \le l \le c}$ and $\{r_j\} = \psi_{k_2}(l)_{1 \le l \le c}$

$\xrightarrow{c,\ k_1,\ k_2}$

4. Uses k_1 and k_2 to generate the set Q
5. Computes $\sigma = \prod_{(j,r_j)\in Q} \sigma_j^{r_j} \in \mathbb{G}_1$
6. Computes $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n,\\ 1\le k\le s}}$, $\mu_{ik} = \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p$

$\xleftarrow{\sigma,\ \mu=\{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}}$

7. Checks $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y)$

The correctness of the above verification equation can be shown as follows:

$$
\begin{aligned}
\hat{e}(\sigma, g) &= \hat{e}(\prod_{(j,r_j)\in Q} \sigma_j^{r_j}, g) = \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} \sigma_{ij}]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j} \cdot \prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \prod_{k=1}^{s} u_k^{r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y)
\end{aligned}
$$

Fig. 3. Challenge response protocol in the MB-PMDDP scheme.

a challenge vector to the CSP and verify the response. Public verifiability can solve disputes that may occur between the data owner and the CSP regarding data integrity. If such a dispute occurs, a trusted third party auditor (TPA) can determine whether the data integrity is maintained or not. Since the owner's public key is only needed to perform the verification step, the owner is not required to reveal his secret key to the TPA. The security analysis of the MB-PMDDP scheme is given in Appendix B (included in accompanying supplementary materials).

IV. Reference Model and Performance Analysis

A. Reference Model

It is possible to obtain a provable multi-copy dynamic data possession scheme by extending existing PDP models for single-copy dynamic data. Such PDP schemes selected for extension must meet the following conditions: (i) support of *full* dynamic operations (modify, insert, append, and delete), (ii) support of public verifiability, (iii) based on pairing cryptography in creating block tags (homomorphic authenticators); and (iv) block tags are outsourced along with data blocks to the CSP (*i.e.*, tags are not stored on the local storage of the data owner). Meeting these conditions allows us to construct a PDP reference model that has similar features to the proposed MB-PMDDP scheme. Therefore, we can establish a fair comparison between the two schemes and evaluate the performance of our proposed approach.

Below we drive a scheme by extending PDP models, which are based on authenticated data structures, see [12], [13]. Using Merkle hash trees (MHTs) [27], we construct a scheme labelled as TB-PMDDP (tree-based provable multi-copy dynamic data possession), but it can also be designed using authenticated skip lists [12] or other authenticated data structures. The TB-PMDDP is used as a reference model for comparing the proposed MB-PMDDP scheme.

1) Merkle Hash Tree: An MHT [27] is a binary tree structure used to efficiently verify the integrity of the data. The MHT is a tree of hashes where the leaves of the tree are the hashes of the data blocks. Let h denotes a cryptographic hash function (*e.g.*, SHA-2), Fig. 4a shows an example of an MHT used for verifying the integrity of a file F consisting of 8 blocks. $h_j = h(b_j)$ $(1 \le j \le 8)$. $h_A = h(h_1||h_2)$, $h_B = h(h_3||h_4)$, and so on. Finally, $h_R = h(h_E||h_F)$ is the hash of the root node that is used to authenticate the integrity

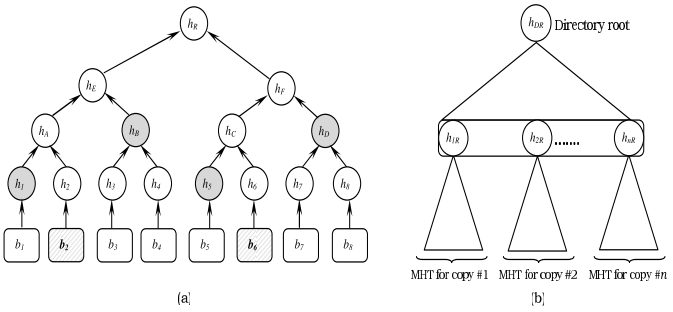

Fig. 4. Hashing trees for outsourced data. (a) Merkle Hash Tree. (b) Directory Tree.

of all data blocks. The data blocks $\{b_1, b_2, \ldots, b_8\}$ are stored on a remote server, and only the authentic value h_R is stored locally on the verifier side. For example, if the verifier requests to check the integrity of the blocks b_2 and b_6, the server will send these two blocks along with the authentication paths $\mathbb{A}_2 = \{h_1, h_B\}$ and $\mathbb{A}_6 = \{h_5, h_D\}$ that are used to reconstruct the root of the MHT. $\mathbb{A}_j$ — the authentication path of b_j — is a set of node siblings (grey-shaded circles) on the path from h_j to the root of the MHT. The verifier uses the received blocks and the authentication paths to recompute the root in the following manner. The verifier constructs $h_2 = h(b_2)$, $h_6 = h(b_6)$, $h_A = h(h_1||h_2)$, $h_C = h(h_5||h_6)$, $h_E = h(h_A||h_B)$, $h_F = h(h_C||h_D)$, and $h_R = h(h_E||h_F)$. After computing h_R, it is compared with the authentic value stored locally on the verifier side.

The MHT is commonly used to authenticate the *values* of the data blocks. In the dynamic behavior of outsourced data, we need to authenticate both the *values* and the *positions* of the data blocks, *i.e.*, we need an assurance that a specific value is stored at a specific leaf node. For example, if a data owner requires to insert a new block after position j, the verifier needs to make sure that the server has inserted the new block at the requested position. To validate the positions of the blocks, the leaf nodes of the MHT are treated in a specific sequence, *e.g.*, left-to-right sequence [28]. So, the hash of any internal node $= h$(left child || right child), *e.g.*, $h_A = h(h_1||h_2) \neq h(h_2||h_1)$. Besides, the authentication path $\mathbb{A}_j$ is viewed as an *ordered* set, and thus any leaf node is uniquely specified by following the used sequence of constructing the root of the MHT.

2) Directory MHT for File Copies: In the TB-PMDDP scheme an MHT is constructed for each file copy, and then the roots of the individual trees are used to build a hash tree which we call a directory MHT. The key idea is to make the root node of each copy's MHT as a leaf node in a directory MHT used to authenticate the integrity of all file copies in a hierarchical manner. The directory tree is depicted in Fig. 4b. The verifier can keep only one hash value (metadata) $\mathcal{M} = h(ID_F||h_{DR})$, where ID_F is a unique file identifier for a file F, and h_{DR} is the authenticated directory root value that can be used to periodically check the integrity of all file copies. Appendix C contains the procedural steps of the derived TB-PMDDP scheme.

TABLE I
NOTATION OF CRYPTOGRAPHIC OPERATIONS

Notation	Description	Notation	Description
h_{SHA}	Cryptographic hashing	$\mathcal{M}_{\mathbb{Z}_p}$	Multiplication in $\mathbb{Z}_p$
$\mathcal{H}_{\mathbb{G}}$	Hashing to $\mathbb{G}$	$\mathcal{A}_{\mathbb{Z}_p}$	Addition in $\mathbb{Z}_p$
$\mathcal{E}_{\mathbb{G}}$	Exponentiation in $\mathbb{G}$	$\mathcal{P}$	Bilinear pairing
$\mathcal{M}_{\mathbb{G}}$	Multiplication in $\mathbb{G}$	E_K	Encryption using K

B. Performance Analysis

Here we evaluate the performance of the presented schemes: MB-PMDDP and TB-PMDDP. The file F used in our performance analysis is of size 64MB with 4KB block size. Without loss of generality, we assume that the desired security level is 128-bit. Thus, we utilize an elliptic curve defined over Galois field $GF(p)$ with $|p| = 256$ bits (a point on this curve can be represented by 257 bits using compressed representation [29]), and a cryptographic hash of size 256 bits (*e.g.*, SHA-256).

Similar to [5], [10], and [17] the computation cost is estimated in terms of the used crypto-operations which are notated in Table I. $\mathbb{G}$ indicates a group of points over a suitable elliptic curve in the bilinear pairing.

Let n, m, and s denote the number of copies, the number of blocks per copy, and the number of sectors per block, respectively. Let c denotes the number of blocks to be challenged, and $|F|$ denotes the size of the file copy. Let the keys used with the PRP and the PRF be of size 128 bits. Table II presents a theoretical analysis for the setup, storage, communication, computation, and dynamic operations costs of the two schemes: MB-PMDDP and TB-PMDDP.

C. Comments

1) System Setup: Table II shows that the setup cost of the MB-PMDDP scheme is less than that of the TB-PMDDP scheme. The TB-PMDDP scheme takes some extra cryptographic hash operations to prepare the MHTs for the file copies to generate the metadata $\mathcal{M}$.

2) Storage Overhead: The storage overhead on the CSP for the MB-PMDDP scheme is much less than that of the TB-PMDDP model. Storage overhead is the additional space used to store some information other than the outsourced file copies $\widetilde{\mathbb{F}}$ ($n|F|$ bits are used to store $\widetilde{\mathbb{F}}$). Both schemes need some additional space to store the aggregated block tags $\Phi = \{\sigma_j\}_{1 \leq j \leq m}$, where σ_j is a group element that can be represented by 257 bits. Besides Φ, the TB-PMDDP scheme needs to store an MHT for each file copy which costs additional storage space on the cloud servers. The MHTs can be computed on the fly during the operations of the TB-PMDDP scheme. This slight modification can reduce the storage overhead on the remote servers, but it will negatively affect the overall system performance. The MHTs are needed through each dynamic operation of the file blocks and through the verification phase of the system. Thus, being not explicitly stored on the CSP can influence the system performance. The CSP storage overhead of the MB-PMDDP scheme is independent of the number of copies n, while it is linear in n for the TB-PMDDP scheme. For 20 copies of the file F,

the overheads on the CSP are 0.50MB and 20.50MB for the MB-PMDDP and TB-PMDDP schemes, respectively (about 97% reduction). Reducing the storage overhead on the CSP side is economically a key feature to reduce the fees paid by the customers.

On the other hand, the MB-PMDDP scheme keeps a map-version table on the verifier side compared with $\mathcal{M}$ (one hash value) for the TB-PMDDP. An entry of the map-version table is of size 8 bytes (two integers), and the total number of entries equals to the number of file blocks. It is important to note that during implementation the $\mathcal{SN}$ is not needed to be stored in the table; $\mathcal{SN}$ is considered to be the entry/table index (the map-version table is implemented as a linked list). Moreover, there is only *one* table for all file copies which mitigates the storage overhead on the verifier side. The size of the map-version table for the file F is only 128KB for unlimited number of copies.

3) Communication Cost: From Table II, the communication cost of the MB-PMDDP scheme is much less than that of the TB-PMDDP. During the response phase, the map-based scheme sends one element σ (257 bits) and $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$, where μ_{ik} is represented by 256 bits. On the other hand, the tree-based approach sends $\langle \sigma, \mu, \{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1\le i\le n\\ (j,*)\in Q}}, \langle \mathbb{A}_{ij} \rangle_{\substack{1\le i\le n\\ (j,*)\in Q}} \rangle$, where each $\mathcal{H}(\tilde{b}_{ij})$ is represented by 257 bits, and $\mathbb{A}_{ij}$ is an authentication path of length $O(\log_2 m)$. Each node along $\mathbb{A}_{ij}$ is a cryptographic hash of size 256 bits. The response of the MB-PMDDP scheme for 20 copies of F is 0.078MB, while it is 4.29MB for the TB-PMDDP (about 98% reduction). The challenge for both schemes is about 34 bytes.

4) Computation Cost: For the two schemes, the cost is estimated in terms of the crypto-operations (see Table I) needed to generate the proof $\mathbb{P}$ and check the verification equation that validates $\mathbb{P}$. As observed from Table II, the cost expression of the proof for the MB-PMDDP scheme has two terms linear in the number of copies n, while the TB-PMDDP scheme has three terms linear in n. Moreover, the MB-PMDDP scheme contains only one term linear in n in the verification cost expression, while there are three terms linear in n in the verification cost expression for the TB-PMDDP scheme. These terms affect the total computation time when dealing with a large number of copies in practical applications.

5) Dynamic Operations Cost: Table II also presents the cost of dynamic operations for both schemes. The communication cost of the MB-PMDDP scheme due to dynamic operations is less than that of the TB-PMDDP scheme for the owner sends a request $\langle IID_F, BlockOp, j, \{b_i^*\}_{1\le i\le n}, \sigma_j^* \rangle$ to the CSP and receives no information back. During the dynamic operations of the TB-PMDDP scheme, the owner sends a request to the CSP and receives the authentication paths which are of order $O(n \log_2(m))$. The authentication paths for updating 20 copies of F equals 8.75KB.

The owner in both schemes uses $n\,E_K$ operations to create the distinct blocks $\{b_i^*\}_{1\le i\le n}$, and $(s+1)n\,\mathcal{E}_{\mathbb{G}} + n\,\mathcal{H}_{\mathbb{G}} + (sn+n-1)\,\mathcal{M}_{\mathbb{G}}$ to generate the aggregated tag σ_j^* (the delete operation does not require this computations). For the MB-PMDDP scheme, the owner updates the state (the map-version table) without usage of cryptographic operations (add, remove, or modify a table entry). On the other hand, updating the state (MHTs on the CSP and $\mathcal{M}$ on the owner) of the TB-PMDDP scheme costs $n\,\mathcal{H}_{\mathbb{G}} + (2n\log_2(m) + 3n)\,h_{SHA}$ to update the MHTs of the file copies according to the required dynamic operations, and regenerate the new directory root that constructs a new $\mathcal{M}$. The experimental results show that updating the state of the TB-PMDDP scheme has insignificant effect on the total computation time of the dynamic operations.

V. Implementation and Experimental Evaluation

A. Implementation

We have implemented the proposed MB-PMDDP scheme and the TB-PMDDP reference model on top of Amazon Elastic Compute Cloud (Amazon EC2) [30] and Amazon Simple Storage Service (Amazon S3) [31] cloud platforms. Through Amazon EC2 customers can lunch and manage Linux/Unix/Windows server instances (virtual servers) in Amazon's infrastructure. The number of EC2 instances can be automatically scaled up and down according to customers' needs. Amazon S3 is a web storage service to store and retrieve almost unlimited amount of data. Moreover, it enables customers to specify geographic locations for storing their data.

Our implementation of the presented schemes consists of three modules: `OModule` (owner module), `CModule` (CSP module), and `VModule` (verifier module). `OModule`, which runs on the owner side, is a library that includes KeyGen, CopyGen, TagGen, and PrepareUpdate algorithms. `CModule` is a library that runs on Amazon EC2 and includes ExecuteUpdate and Prove algorithms. `VModule` is a library to be run at the verifier side and includes the Verify algorithm.

In the experiments, we do not consider the system pre-processing time to prepare the different file copies and generate the tags set. This pre-processing is done only once during the life time of the system which may be for tens of years. Moreover, in the implementation we do not consider the time to access the file blocks, as the state-of-the-art hard drive technology allows as much as 1MB to be read in just few nanoseconds [5]. Hence, the total access time is unlikely to have substantial impact on the overall system performance.

1) Implementation Settings: A "large" Amazon EC2 instance is used to run `CModule`. Through this instance, a customers gets total memory of size 7.5GB and 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each). One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0–1.2GHz 2007 Opteron or 2007 Xeon processor [32]. The `OModule` and `VModule` are executed on a desktop computer with Intel(R) Xeon(R) 2GHz processor and 3GB RAM running Windows XP. We outsource copies of a data file of size 64MB to Amazon S3. Algorithms (encryption, pairing, hashing, etc.) are implemented using MIRACL library version 5.4.2. For 128-bit security level, the elliptic curve group we work on has a 256-bit group order. In the experiments, we utilize the Barreto-Naehrig (BN) [33] curve defined over prime field $GF(p)$ with $|p| = 256$ bits and embedding degree $= 12$

TABLE II
PERFORMANCE OF THE MB-PMDDP AND TB-PMDDP SCHEMES

Costs		MB-PMDDP	TB-PMDDP
System Setup	Tag Generation	$(s+1)nm\,\mathcal{E}_{\mathbb{G}} + nm\,\mathcal{H}_{\mathbb{G}} + (sn+n-1)m\,\mathcal{M}_{\mathbb{G}}$	$(s+1)nm\,\mathcal{E}_{\mathbb{G}} + nm\,\mathcal{H}_{\mathbb{G}} + (sn+n-1)m\,\mathcal{M}_{\mathbb{G}}$
	Metadata Generation	—	$2nm\,h_{SHA}$
Storage	File Copies	$n\|F\|$	$n\|F\|$
	CSP Overhead	$257m$ bits	$257m$ + $(512m$ - $256)n$ bits
	Verifier Overhead	$64m$ bits	256 bits
Communication	Challenge	256 + $\log_2(c)$ *bits*	256 + $\log_2(c)$ *bits*
	Response	257 + $256sn$ bits	257 + $256sn$ + $(256\log_2(m)$ + $257)cn$ bits ‡
Computation	Proof	$c\,\mathcal{E}_{\mathbb{G}} + (c-1)\,\mathcal{M}_{\mathbb{G}} + csn\,\mathcal{M}_{\mathbb{Z}_p} + (c-1)sn\,\mathcal{A}_{\mathbb{Z}_p}$	$c\,\mathcal{E}_{\mathbb{G}} + (c-1)\,\mathcal{M}_{\mathbb{G}} + csn\,\mathcal{M}_{\mathbb{Z}_p} + (c-1)sn\,\mathcal{A}_{\mathbb{Z}_p} + cn\,\mathcal{H}_{\mathbb{G}}$
	Verification	$2\mathcal{P} + (c+s+1)\,\mathcal{E}_{\mathbb{G}} + c\,\mathcal{H}_{\mathbb{G}} + (c+s-1)\,\mathcal{M}_{\mathbb{G}} + s(n-1)\mathcal{A}_{\mathbb{Z}_p}$	$(c\log_2(m)+2)n\,h_{SHA}$ ‡ + $2\mathcal{P}$ + $(c+s)\,\mathcal{E}_{\mathbb{G}} + (cn+s-1)\,\mathcal{M}_{\mathbb{G}} + s(n-1)\,\mathcal{A}_{\mathbb{Z}_p}$
Dynamic Operations	Communication	"Request"	"Request" + $O(n\log_2(m))$
	Owner Computation (Modify/Insert/Append)	$n\,E_K + (s+1)n\,\mathcal{E}_{\mathbb{G}} + n\,\mathcal{H}_{\mathbb{G}} + (sn+n-1)\,\mathcal{M}_{\mathbb{G}}$	$n\,E_K + (s+1)n\,\mathcal{E}_{\mathbb{G}} + n\,\mathcal{H}_{\mathbb{G}} + (sn+n-1)\,\mathcal{M}_{\mathbb{G}}$
	State Update	—	$n\,\mathcal{H}_{\mathbb{G}} + (2n\log_2(m)+3n)\,h_{SHA}$

‡ $\log_2(m)$ is the upper bound of the authentication path length when $c > 1$.

(the BN curve with these parameters is provided by the MIRACL library).

B. Experimental Evaluation

We compare the presented two schemes from different perspectives: proof computation times, verification times, and cost of dynamic operations. It has been reported in [1] that if the remote server is missing a fraction of the data, then the number of blocks that needs to be checked in order to detect server misbehavior with high probability is constant independent of the total number of file blocks. For example, if the server deletes 1% of the data file, the verifier only needs to check for $c = 460$-randomly chosen blocks of the file so as to detect this misbehavior with probability larger than 99%. Therefore, in our experiments, we use $c = 460$ to achieve a high probability of assurance.

1) Proof Computation Time: For different number of copies, Fig. 5a presents the proof computation times (in seconds) to provide an evidence that the file copies are actually stored on the cloud servers in an updated, uncorrupted, and consistent state. The timing curve of the MB-PMDDP scheme is much less than that of the TB-PMDDP. For 20 copies, the proof computation times for the MB-PMDDP and the TB-PMDDP schemes are 1.51 and 5.58 seconds, respectively ($\approx$ 73% reduction in the computation time). As observed from Fig. 5a, the timing curve of the TB-PMDDP scheme grows with increasing number of copies at a rate higher than that of the MB-PMDDP. That is because the proof cost expression of the TB-PMDDP scheme contains more terms which are linear in the number of copies n (Table II).

2) Verification Time: Fig. 5b presents the verification times (in seconds) to check the responses/proofs received from the CSP. The MB-PMDDP scheme has verification times less than that of the TB-PMDDP scheme. For 20 copies, the verification times for the MB-PMDDP and the TB-PMDDP schemes are 1.58 and 3.13 seconds, respectively (about 49% reduction in the verification time). The verification timing curve of the MB-PMDDP scheme is *almost* constant. There is a very small increase in the verification time with increasing number of copies. This is due to the fact that although the term $s(n-1)\mathcal{A}_{\mathbb{Z}_p}$ in the verification cost of the MB-PMDDP scheme is linear in n (Table II), in our experiments its numerical value is quite small compared to those of the other terms in the cost expression. This feature makes the the MB-PMDDP scheme computationally cost-effective and more efficient when verifying a large number of file copies.

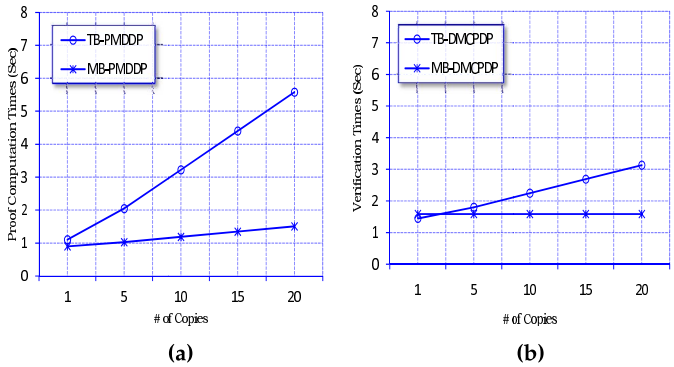

Fig. 5. Computation costs of the MB-PMDDP and TB-PMDDP schemes. (a) CSP computation times (sec). (b) Verifier computation times (sec).

TABLE III
OWNER COMPUTATION TIMES (SEC) DUE TO DYNAMIC OPERATIONS ON A SINGLE BLOCK

# of Copies	1	5	10	15	20
MB-PMDDP	0.261	1.304	2.608	3.913	5.217
TB-PMDDP	0.261	1.305	2.610	3.916	5.221

3) Dynamic Operations Cost: For different number of copies, Table III presents the computation times (in seconds) on the owner side of the two schemes due to dynamic opera-

Algorithm 1 BS(σList, μList, start, end)

```
begin
  len ⟵ (end−start)+1            /* The list length */
  if len = 1 then
    σ ⟵ σList[start]
    {μ_k}_{1≤k≤s} ⟵ μList[start][k]
    ê(σ, g) ?= ê(∏_{(j,r_j)∈Q} H(ID_F||BN_j||BV_j)^{r_j} · ∏_{k=1}^{s} u_k^{μ_k}, y)
    if NOT verified then
      invalidList.Add(start)
    end
  else
    σ ⟵ ∏_{i=1}^{len} σList[start+i − 1]
    {μ_ik}_{1≤i≤len, 1≤k≤s} ⟵ μList[start+i − 1][k]
    ê(σ, g) ?= ê([∏_{(j,r_j)∈Q} H(ID_F||BN_j||BV_j)^{r_j}]^{len} · ∏_{k=1}^{s} u_k^{∑_{i=1}^{len} μ_ik}, y)
    if NOT verified then
      /* work with the left and right halves of
      σList and μList */
      mid ⟵ ⌊(start+end)/2⌋   /* List middle */
      BS(σList, μList, start, mid)   /* Left part */
      BS(σList, μList, mid+1, end)   /* Right */
    end
  end
end
```

tions on a single block. The owner computation times for both schemes are *approximately* equal. The slight increase of the TB-PMDDP scheme is due to some additional hash operations required to regenerate a new directory root that constructs a new $\mathcal{M}$ (Table II). As noted, the computation overhead on the owner side is practical. It takes about 5 seconds to modify/insert/append a block of size 4KB on 20 copies ($<$ 1 minute for 200 copies). In the experiments, we use only *one* desktop computer to accomplish the organization (data owner) work. In practice during updating the outsourced copies, the owner may choose to split the work among a few devices inside the organization or use a single device with a *multi-core* processor which is becoming prevalent these days, and thus the computation time on the owner side is significantly reduced in many applications.

VI. Identifying Corrupted Copies

Here, we show how the proposed MB-PMDDP scheme can be slightly modified to identify the indices of corrupted copies. The proof $\mathbb{P} = \{\sigma, \mu\}$ generated by the CSP will be valid and will pass the verification equation (1) only if all copies are intact and consistent. Thus, when there is one or more corrupted copies, the whole auditing procedure fails. To handle this situation and identify the corrupted copies, a slightly modified version of the MB-PMDDP scheme can be used. In this version, the data owner generates a tag σ_{ij} for each block $\tilde{b}_{ij}$, but does not aggregate the tags for the blocks at the same indices in each copy, *i.e.*, $\Phi = \{\sigma_{ij}\}_{\substack{1\le i\le n\\ 1\le j\le m}}$. During the response phase, the CSP computes $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$ as before, but $\sigma = \prod_{(j,r_j)\in Q}[\prod_{i=1}^{n}\sigma_{ij}]^{r_j} \in \mathbb{G}_1$. Upon receiving the proof $\mathbb{P} = \{\sigma, \mu\}$, the verifier first validates $\mathbb{P}$ using equation (1). If the verification fails, the verifier asks the CSP to send $\sigma = \{\sigma_i\}_{1\le i\le n}$, where $\sigma_i = \prod_{(j,r_j)\in Q}\sigma_{ij}^{r_j}$. Thus, the verifier has two lists σList $= \{\sigma_i\}_{1\le i\le n}$ and μList $= \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$ (μList is a two dimensional list).

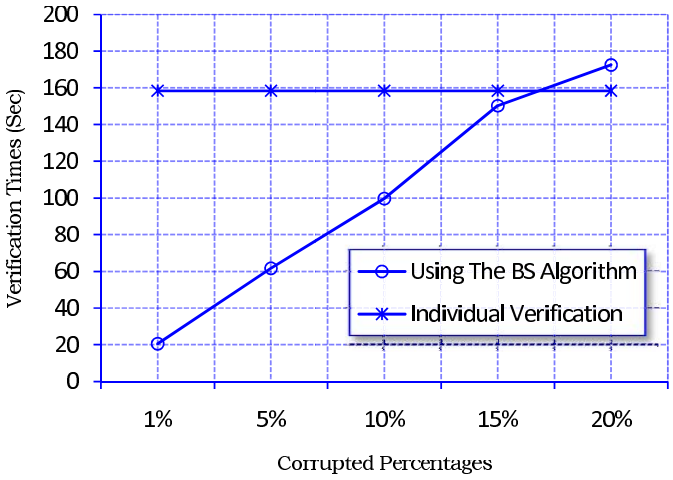

Fig. 6. Verification times with different percentages of corrupted copies.

Utilizing a recursive divide-and-conquer approach (binary search) [34], the verifier can identify the indices of corrupted copies. Specifically, σList and μList are divided into halves: σList$\rightarrow$ (σLeft:σRight), and μList$\rightarrow$ (μLeft:μRight). The verification equation (1) is applied recursively on σLeft with μLeft and σRight with μRight. Note that the individual tags in σLeft or σRight are aggregated via multiplication to generate one σ that is used during the recursive application of equation (1). The procedural steps of identifying the indices of corrupted copies are indicated in Algorithm 1.

The BS (binary search) algorithm takes four parameters: σList, μList, start that indicates the start index of the currently working lists, and end to indicate the last index of these lists. The initial call to the BS algorithm takes (σList, μList, 1, n). The invalid indices are stored in invalidList (a global data structure).

This slight modification to identify the corrupted copies will be associated with some extra storage overhead on the cloud servers, where the CSP has to store mn tags for the file copies $\widetilde{\mathbb{F}}$ (m tags in the original version). Moreover, the challenge-response phase may be done in two rounds if the initial round to verify all copies fails.

We design experiments (using same file/parameters from Section V) to show the effect of identifying the corrupted copies on the verification time. We generate 100 copies, which are verified in 1.584 seconds when all copies are accurate. A percentage — ranging from 1% to 20% — of the file copies is *randomly* corrupt. Fig. 6 shows the verification time (in seconds) with different corrupted percentages. The verification time is about 20.58 seconds when 1% of

the copies are invalid. As observed from Fig. 6, when the percentages of corrupted copies are up to 15% of the total copies, the performance of using the BS algorithm in the verification is more efficient than individual verification for each copy. It takes about 1.58 seconds to verify one copy, and thus individual verifications of 100 copies requires $100 \times 1.58 = 158$ seconds.

In short, the proposed scheme can be slightly modified to support the feature of identifying the corrupted copies at the cost of some extra storage/communication/computation overheads. For the CSP to remain in business and maintain a good reputation, invalid responses to verifier's challenges are sent in very rare situations, and thus the original version of the proposed scheme is used in most of the time.

VII. Summary and Concluding Remarks

Outsourcing data to remote servers has become a growing trend for many organizations to alleviate the burden of local data storage and maintenance. In this work we have studied the problem of creating multiple copies of dynamic data file and verifying those copies stored on untrusted cloud servers.

We have proposed a new PDP scheme (referred to as MB-PMDDP), which supports outsourcing of multi-copy dynamic data, where the data owner is capable of not only archiving and accessing the data copies stored by the CSP, but also updating and scaling these copies on the remote servers. To the best of our knowledge, the proposed scheme is the first to address *multiple* copies of *dynamic* data. The interaction between the authorized users and the CSP is considered in our scheme, where the authorized users can seamlessly access a data copy received from the CSP using a single secret key shared with the data owner. Moreover, the proposed scheme supports public verifiability, enables arbitrary number of auditing, and allows *possession-free* verification where the verifier has the ability to verify the data integrity even though he neither possesses nor retrieves the file blocks from the server.

Through performance analysis and experimental results, we have demonstrated that the proposed MB-PMDDP scheme outperforms the TB-PMDDP approach derived from a class of dynamic single-copy PDP models. The TB-PMDDP leads to high storage overhead on the remote servers and high computations on both the CSP and the verifier sides. The MB-PMDDP scheme significantly reduces the computation time during the challenge-response phase which makes it more practical for applications where a large number of verifiers are connected to the CSP causing a huge computation overhead on the servers. Besides, it has lower storage overhead on the CSP, and thus reduces the fees paid by the cloud customers. The dynamic block operations of the map-based approach are done with less communication cost than that of the tree-based approach.

A slight modification can be done on the proposed scheme to support the feature of identifying the indices of corrupted copies. The corrupted data copy can be reconstructed even from a complete damage using duplicated copies on other servers. Through security analysis, we have shown that the proposed scheme is provably secure.

References

[1] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2007, pp. 598–609.

[2] K. Zeng, "Publicly verifiable remote data integrity," in *Proc. 10th Int. Conf. Inf. Commun. Secur. (ICICS)*, 2008, pp. 419–434.

[3] Y. Deswarte, J.-J. Quisquater, and A. Saïdane, "Remote integrity checking," in *Proc. 6th Working Conf. Integr. Internal Control Inf. Syst. (IICIS)*, 2003, pp. 1–11.

[4] D. L. G. Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer," IACR (International Association for Cryptologic Research) ePrint Archive, Tech. Rep. 2006/150, 2006.

[5] F. Sebé, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 8, pp. 1034–1038, Aug. 2008.

[6] P. Golle, S. Jarecki, and I. Mironov, "Cryptographic primitives enforcing communication and storage complexity," in *Proc. 6th Int. Conf. Financial Cryptograph. (FC)*, Berlin, Germany, 2003, pp. 120–135.

[7] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in *Proc. 11th USENIX Workshop Hot Topics Oper. Syst. (HOTOS)*, Berkeley, CA, USA, 2007, pp. 1–6.

[8] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," IACR Cryptology ePrint Archive, Tech. Rep. 2008/186, 2008.

[9] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *ACM Trans. Storage*, vol. 2, no. 2, pp. 107–138, 2006.

[10] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. 4th Int. Conf. Secur. Privacy Commun. Netw. (SecureComm)*, New York, NY, USA, 2008, Art. ID 9.

[11] C. Wang, Q. Wang, K. Ren, and W. Lou. (2009). "Ensuring data storage security in cloud computing," IACR Cryptology ePrint Archive, Tech. Rep. 2009/081. [Online]. Available: http://eprint.iacr.org/

[12] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2009, pp. 213–222.

[13] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proc. 14th Eur. Symp. Res. Comput. Secur. (ESORICS)*, Berlin, Germany, 2009, pp. 355–370.

[14] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 9, pp. 1432–1437, Sep. 2011.

[15] A. F. Barsoum and M. A. Hasan. (2010). "Provable possession and replication of data over cloud servers," Centre Appl. Cryptograph. Res., Univ. Waterloo, Waterloo, ON, USA, Tech. Rep. 2010/32. [Online]. Available: http://www.cacr.math.uwaterloo.ca/techreports/2010/cacr2010-32.pdf

[16] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Proc. 28th IEEE ICDCS*, Jun. 2008, pp. 411–420.

[17] Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in *Proc. 2nd Int. Symp. Data, Privacy, E-Commerce*, Sep. 2010, pp. 84–89.

[18] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. 14th Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2008, pp. 90–107.

[19] A. Juels and B. S. Kaliski, Jr., "Pors: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 584–597.

[20] R. Curtmola, O. Khan, and R. Burns, "Robust remote data checking," in *Proc. 4th ACM Int. Workshop Storage Secur. Survivability*, 2008, pp. 63–68.

[21] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in *Proc. ACM Workshop Cloud Comput. Secur. (CCSW)*, 2009, pp. 43–54.

[22] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Proc. 6th Theory Cryptograph. Conf. (TCC)*, 2009, pp. 109–127.

[23] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2009, pp. 187–198.
[24] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, 1949.
[25] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Proc. 7th Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, London, U.K., 2001, pp. 514–532.
[26] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *Proc. 15th Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, Berlin, Germany, 2009, pp. 319–333.
[27] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. IEEE Symp. Secur. Privacy*, Apr. 1980, p. 122.
[28] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, no. 1, pp. 21–41, Jan. 2004.
[29] P. S. L. M. Barreto and M. Naehrig, *Pairing-Friendly Elliptic Curves of Prime Order With Embedding Degree 12*, IEEE Standard P1363.3, 2006.
[30] *Amazon Elastic Compute Cloud (Amazon EC2)*. [Online]. Available: http://aws.amazon.com/ec2/, accessed Aug. 2013.
[31] *Amazon Simple Storage Service (Amazon S3)*. [Online]. Available: http://aws.amazon.com/s3/, accessed Aug. 2013.
[32] *Amazon EC2 Instance Types*. [Online]. Available: http://aws.amazon.com/ec2/, accessed Aug. 2013.
[33] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proc. 12th Int. Workshop SAC*, 2005, pp. 319–331.
[34] A. L. Ferrara, M. Green, S. Hohenberger, and M. Ø. Pedersen, "Practical short signature batch verification," in *Proc. Cryptograph. Track RSA Conf.*, 2009, pp. 309–324.
[35] A. F. Barsoum and M. A. Hasan. (2011). "On verifying dynamic multiple data copies over cloud servers," IACR Cryptology ePrint Archive, Tech. Rep. 2011/447. [Online]. Available: http://eprint.iacr.org/
[36] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Efficient provable data possession for hybrid clouds," in *Proc. 17th ACM Conf. Comput. Commun. Secur. (CCS)*, 2010, pp. 756–758.

Ayad F. Barsoum is currently an Assistant Professor with the Department of Computer Science, St. Mary's University, San Antonio, TX, USA. He received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo (UW), Waterloo, ON, Canada, in 2013, where he is a member of the Centre for Applied Cryptographic Research.

He has received the Graduate Research Studentship, the International Doctoral Award, and the University of Waterloo Graduate Scholarship at UW. He received the B.Sc. and M.Sc. degrees in computer science from Ain Shams University, Cairo, Egypt.

M. Anwar Hasan received the B.Sc. degree in electrical and electronic engineering and the M.Sc. degree in computer engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 1986 and 1988, respectively, and the Ph.D. degree in electrical engineering from the University of Victoria, Victoria, BC, Canada, in 1992.

He joined the Department of Electrical and Computer Engineering, University of Waterloo (UW), Waterloo, ON, Canada, in 1993, where he has been a Full Professor since 2002. At UW, he is currently a member of the Centre for Applied Cryptographic Research, the Center for Wireless Communications, and the VLSI Research Group.