Enabling Dynamic Data and Indirect Mutual Trust for Cloud Computing Storage Systems

Ayad F. Barsoum and M. Anwar Hasan
Department of Electrical and Computer Engineering,
University of Waterloo, Ontario, Canada.

Abstract

Currently, the amount of *sensitive* data produced by many organizations is outpacing their storage ability. The management of such huge amount of data is quite expensive due to the requirements of high storage capacity and qualified personnel. Storage-as-a-Service (SaaS) offered by cloud service providers (CSPs) is a *paid* facility that enables organizations to outsource their data to be stored on remote servers. Thus, SaaS reduces the maintenance cost and mitigates the burden of large local data storage at the organization's end. A data owner pays for a desired level of security and must get some compensation in case of any misbehavior committed by the CSP. On the other hand, the CSP needs a protection from any false accusation that may be claimed by the owner to get illegal compensations.

In this paper, we propose a cloud-based storage scheme that allows the data owner to benefit from the facilities offered by the CSP and enables *indirect* mutual trust between them. The proposed scheme has four important features: (i) it allows the owner to outsource sensitive data to a CSP, and perform full block-level dynamic operations on the outsourced data, i.e., block modification, insertion, deletion, and append, (ii) it ensures that authorized users (i.e., those who have the right to access the owner's file) receive the latest version of the outsourced data, (iii) it enables indirect mutual trust between the owner and the CSP, and (iv) it allows the owner to grant or revoke access to the outsourced data. We discuss the security issues of the proposed scheme. Besides, we justify its performance through theoretical analysis and a prototype implementation on Amazon cloud platform to evaluate storage, communication, and computation overheads.

Index Terms

Outsourcing data storage, dynamic environment, mutual trust, access control

1 Introduction

Cloud computing has received considerable attention from both academia and industry due to a number of important advantages including: cost effectiveness, low management overhead, immediate access to a wide range of applications, flexibility to scale up and down information technology (IT) capacity, and mobility where customers can access information wherever they are, rather than having to remain at their desks. Cloud computing is a distributed computational model over a large pool of shared-virtualized

computing resources (e.g., storage, processing power, memory, applications, services, and network bandwidth). Cloud service providers (CSPs) offer different classes of services (Storage-as-a-Service (SaaS), Application-as-a-Service, and Platform-as-a-Service) that allow organizations to concentrate on their core business and leave the IT operations to experts.

In the current era of digital world, different organizations produce a large amount of sensitive data including personal information, electronic health records, and financial data. The amount of digital data increases at a staggering rate; doubling almost every year and a half [1]. This data needs to be widely distributed and stored for a long time due to operational purposes and regulatory compliance. The local management of such huge amount of data is problematic and costly. While there is an observable drop in the cost of storage hardware, the management of storage has become more complex and represents approximately 75% of the total ownership cost [1]. SaaS offered by CSPs is an emerging solution to mitigate the burden of large local data storage and reduce the maintenance cost via the concept of outsourcing data storage.

Through outsourcing data storage scenario, data owners delegate the storage and management of their data to a CSP in exchange for pre-specified fees metered in GB/month. Such outsourcing of data storage enables owners to store more data on remote servers than on private computer systems. Moreover, the CSP often provides better disaster recovery by replicating the data on multiple servers across multiple data centers achieving a higher level of availability. Thus, many authorized users are allowed to access the remotely stored data from different geographic locations making it more convenient for them.

Since the owner physically releases sensitive data to a remote CSP, there are some concerns regarding *confidentiality*, *integrity*, and *access control* of the data. In some practical applications, data confidentiality is not only a privacy concern, but also a juristic issue. For example, in e-Health applications inside the USA the usage and exposure of protected health information should meet the policies admitted by Health Insurance Portability and Accountability Act (HIPAA) [2], and thus keeping the data private on the remote storage servers is not just an option, but a demand. The confidentiality feature can be guaranteed by the owner via encrypting the data before outsourcing to remote servers.

For verifying data integrity over cloud servers, researchers have proposed provable data possession (PDP) technique to validate the intactness of data stored on remote sites. A number of PDP protocols have been presented to efficiently validate the integrity of *static* data, e.g., [3]–[10]. Another class of PDP schemes was concerned with the *dynamic* behavior of data over remote servers [11]–[15]. This class allows the owner to outsource a data file and perform updating or scaling operations on the outsourced data. Later, a verifier validates that the remote servers keep the data intact and compatible with the dynamic requests issued by the owner. A complementary line of research on PDP has focused on *multiple* data copies stored over different servers [16]–[19]. Proof of retrievability (POR) [20]–[25] was introduced as a stronger technique than PDP in the sense that the entire data file can be reconstructed from portions of the data that are reliably stored on the servers.

Commonly, traditional access control techniques assume the existence of the data owner and the storage servers in the same trust domain. This assumption, however, no longer holds when the data is outsourced to a remote CSP, which takes the full charge of the outsourced data management, and resides outside the trust domain of the data owner. A feasible solution can be presented to enable the owner to enforce access control of the data stored on a remote untrusted CSP. Through this solution, the data is encrypted under a certain key, which is shared only with the authorized users. The unauthorized users, including the CSP, are unable to access the data since they do not have the decryption key. This general solution has been widely incorporated into existing schemes [26]–[29], which aim at providing data storage security on untrusted remote servers. Another class of solutions utilizes attribute-based encryption (ABE) to achieve fine-grained access control. ABE [30][1] is a public key cryptosystem for one-to-many communications that enables fine-grained sharing of encrypted data. The ABE associates the ciphertext with a set of attributes, and the private key with an access structure (policy). The ciphertext is decrypted if and only if the associated attributes satisfy the access structure of the private key. Access revocation in ABE-based systems is an issue since each attribute is conceivably shared by many users. Examples of ABE-based systems for achieving access control of remotely stored data are [32]–[34].

Different approaches have been investigated that encourage the owner to outsource the data, and offer some sort of guarantee related to the confidentiality, integrity, and access control of the outsourced data. These approaches can prevent and detect (with high probability) malicious actions from the CSP side. On the other hand, the CSP needs to be safeguarded from a dishonest owner, who attempts to get illegal compensations by falsely claiming data corruption over cloud servers. This concern, if not properly handled, can cause the CSP to go out of business [35].

In this work, we propose a scheme that addresses some important issues related to outsourcing the storage of data, namely *dynamic data*, *newness*, *mutual trust*, and *access control*. One of the core design principles of data outsourcing is to provide dynamic scalability of data for various applications. This means that the remotely stored data can be not only accessed by authorized users, but also updated and scaled by the owner. After updating, the authorized users should receive the latest version of the data (newness property), i.e., a technique is required to detect whether the received data is stale. This issue is crucial for applications in which critical decisions are taken based on the received data. For example, in e-Health applications a physician may write a prescription based on a patient's medical history received from remote servers. If such medical data is not up-to-date, the given prescription may conflict with the patient's current circumstances causing severe health problems. Mutual trust between the data owner and the CSP is another imperative issue, which is addressed in the proposed scheme. A mechanism is introduced to determine the dishonest party, i.e., misbehavior from any side is detected

1. The construction presented in [30] is called key-policy ABE (KP-ABE), which contrasts with another construction called ciphertext-policy ABE (CP-ABE) [31]. In the CP-ABE, an access structure is associated with the ciphertext, and a set of attributes is associated with the private key.

and the responsible party is identified. Last but not least, the access control is considered, which allows the data owner to grant or revoke access rights to the outsourced data.

Main contributions. Our contributions can be summarized in two main points.

1) The design and implementation of a cloud-based storage scheme that has the following features:
   - It allows a data owner to outsource the data to a remote CSP, and perform full dynamic operations at the block-level, i.e., it supports operations such as block modification, insertion, deletion, and append
   - It ensures the newness property, i.e., the authorized users receive the most recent version of the data
   - It establishes *indirect* mutual trust between the data owner and the CSP since each party resides in a different trust domain
   - It enforces the access control for the outsourced data
2) We discuss the security features of the proposed scheme. Besides, we justify its performance through theoretical analysis and a prototype implementation on Amazon cloud platform to evaluate storage, communication, and computation overheads.

Paper organization. The remainder of the paper is organized as follow. Section 2 contains related work. Our system and assumptions are presented in Section 3. Section 4 reviews some techniques pertaining to our construction. The proposed scheme is elaborated in Section 5. The security analysis is given in Section 6. The performance analysis is shown in Section 7. Section 8 presents the implementation and experimental results. Concluding remarks are given in Section 9.

2 Related Work

Existing research close to our work can be found in the areas of integrity verification of outsourced data, cryptographic file systems in distributed networks, and access control of outsourced data.

Different variations of PDP protocols have been presented for *static* or warehoused data; for example, see [3]–[10]. Some other PDP schemes consider the case of *dynamic* data that are usually more prevailing in practical applications. Examples of PDP schemes that deal with dynamic data are [11]–[15]. While the schemes [3]–[15] are for a *single* copy of a data file, PDP schemes have been presented for multiple copies of *static* data, e.g., [16]–[18]. Reference [19] addresses a PDP construction for *multiple* copies of *dynamic* data. Proof of retrievability (POR) is a complementary approach to PDP, and is stronger than PDP in the sense that the entire data file can be reconstructed from portions of the data that are reliably stored on the servers. This is due to encoding of the data file, for example using *erasure codes*, before outsourcing to remote servers. References [20]–[25] are examples of POR schemes that can be found in the literature.

Kallahalla et al. [26] designed a cryptography-based file system called Plutus for secure sharing of data on untrusted servers. Some authorized users of the data have the privilege to read and write, while

others can only read the data. In Plutus, a file-group represents a set of files with similar attributes, and each file-group is associated with a symmetric key called *file-lockbox* key. A data file is fragmented into blocks, where each block is encrypted with a unique symmetric key called a *file-block* key. The file-block key is further encrypted with the file-lockbox key of the file-group to which the data file belongs. If the data owner wants to share a file-group with a set of users, the file-lockbox key is just distributed to them. Plutus supports two operations on the file blocks: read and write/modify. Delete operation can be supported by overwriting an existing block with *null*.

Goh et al. [27] have presented SiRiUS, which is designed to be layered over existing file systems such as NFS (network file system) to provide end-to-end security. To enforce access control in SiRiUS, each data file (*d-file*) is attached with a metadata file (*md-file*) that contains an encrypted key block for each authorized user with some access rights (read or write). More specifically, the md-file represents the d-file's access control list (ACL). The d-file is encrypted using a file encryption key (FEK), and each entry in the ACL contains an encrypted version of the FEK under the public key of one authorized user. For large-scale sharing, the authors in [27] presented SiRiUS-NNL that uses NNL (Naor-Naor-Lotspiech) broadcast encryption algorithm [36] to encrypt the FEK of each file instead of encrypting using each authorized user's public key. SiRiUS supports two operations on the file blocks: read and write/modify.

Based on proxy re-encryption [37], Ateniese et al. [28] have introduced a secure distributed storage protocol. In their protocol, a data owner encrypts the blocks with symmetric data keys, which are encrypted using a master public key. The owner keeps a master private key to decrypt the symmetric data keys. Using the master private key and the authorized user's public key, the owner generates proxy re-encryption keys. A semi-trusted server then uses the proxy re-encryption keys to translate a ciphertext into a form that can be decrypted only by granted users, and thus enforces access control for the data.

Vimercati et al. [29] have constructed a scheme for securing data on semi-trusted storage servers based on key derivation methods of [38]. In their scheme, a secret key is assigned to each authorized user, and data blocks are grouped based on users that can access these blocks. One key is used to encrypt all blocks in the same group. Moreover, the data owner generates public tokens to be used along with the user's secret key to derive decryption keys of specific blocks. The blocks and the tokens are sent to remote servers, which are not able to drive the decryption key of any block using just the public tokens. The approach in [29] allows the servers to conduct a second level of encryption (over-encryption) to enforce access control of the data. Repeated access grant and revocation may lead to a complicated hierarchy structure for key management [39].

The concept of over-encryption to enforce access control has also been used by Wang et al. [39]. In their scheme, the owner encrypts the data block-by-block, and constructs a binary tree of the block keys. The binary tree enables the owner to reduce the number of keys given to each user, where different keys in the tree can be generated from one common parent node. The remote storage server performs over-encryption to prevent revoked users from getting access to updated data blocks.

Popa et al. [35] have introduced a cryptographic cloud storage system called CloudProof that provides read and write data sharing. CloudProof has been designed to offer security guarantees in the service level agreements of cloud storage systems. It divides the security properties in four categories: confidentiality, integrity, read freshness, and write-serializability. CloudProof can provide these security properties using attestations (signed messages) and chain hash. Besides, it can detect and prove to a third party that any of these properties have been violated. Read freshness and write-serializability in CloudProof are guaranteed by periodic auditing in a *centralized* manner. The time is divided into epochs, which are time periods at the end of each the data owner performs the auditing process. The authorized users send the attestations – they receive from the CSP during the epoch – to the owner for auditing. Like Plutus and SiRiUS, CloudProof supports two operations on the file blocks: read and write/modify.

Discussion. Some aspects related to outsourcing data storage are beyond the setting of both PDP and POR, e.g., enforcing access control, and ensuring the newness of data delivered to authorized users. Even in the case of dynamic PDP, a verifier can validate the correctness of data, but the server is still able to cheat and return stale data to authorized users after the auditing process is done. The schemes [26]–[29] have focused on access control and secure sharing of data on untrusted servers. The issues of full block-level dynamic operations (modify, insert, delete, and append), and achieving mutual trust between the data owners and the remote servers are outside their scope. Although [39] have presented an efficient access control technique and handled full dynamic operations for the data over remote servers, data integrity, newness property, and mutual trust are not addressed. Authorized users in CloudProof [35] are not performing *immediate* checking for freshness of received data; the attestations are sent at the end of each epoch to the owner for completing the auditing task. Instantaneous validation of data freshness is crucial before taking any decisions based on the received data from the cloud. CloudProof guarantees write-serializability, which is outside the scope of our current work as we are focusing on owner-write-users-read applications.

3 Our System and Assumptions

System components and relations. The cloud computing storage model considered in this work consists of four main components as illustrated in Fig. 1: (i) a data owner that can be an organization generating sensitive data to be stored in the cloud and made available for controlled external use; (ii) a CSP who manages cloud servers and provides paid storage space on its infrastructure to store the owner's files and make them available for authorized users; (iii) authorized users – a set of owner's clients who have the right to access the remote data; and (iv) a trusted third party (TTP), an entity who is trusted by all other system components, and has expertise and capabilities to detect and specify dishonest parties.

In Fig. 1, the relations between different system components are represented by *double-sided* arrows, where solid and dashed arrows represent trust and distrust relations, respectively. For example, the data

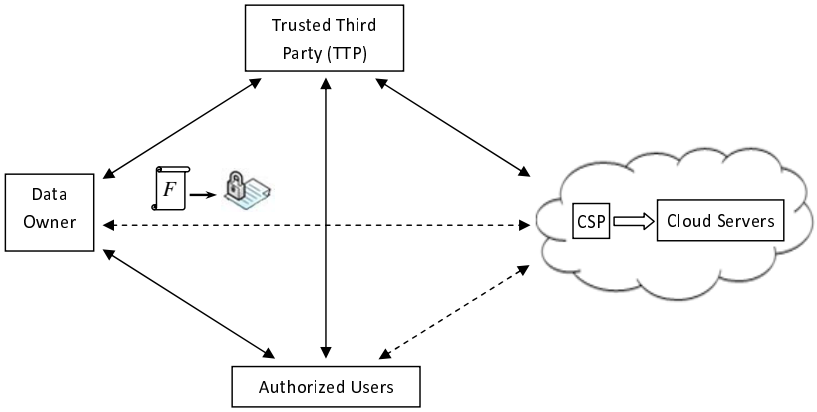

Fig. 1: Cloud computing data storage system model.

owner, the authorized users, and the CSP trust the TTP. On the other hand, the data owner and the authorized users have mutual distrust relations with the CSP. Thus, the TTP is used to enable *indirect* mutual trust between these three components. There is a direct trust relation between the data owner and the authorized users.

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 cloud servers. In these types of applications, a medical center can be considered as the data owner, physicians as the authorized users who have the right to access the patients' medical history, and an independent-trusted organization as the TTP. Many other practical applications like financial, scientific, and educational applications can be viewed in similar settings.

Remark 1. The idea of using a third party auditor has been used before in outsourcing data storage systems, especially for customers with constrained computing resources and capabilities, e.g., [8], [9], [13], [40]. The main focus of a third party auditor is to verify the data stored on remote servers, and give incentives to providers for improving their services. The proposed scheme in this work uses the TTP in a slightly different fashion. The auditing process of the data received from the CSP is done by the authorized users, and we resort to the TTP only to resolve disputes that may arise regarding data integrity or newness. Reducing the storage overhead on the CSP side is economically a key feature to lower the fees paid by the customers. Moreover, decreasing the overall computation cost in the system is another crucial aspect. To achieve these goals, a small part of the owner's work is delegated to the TTP.

Outsourcing, updating, and accessing. The data owner has a file F consisting of m blocks to be outsourced to a CSP, where storage fees are pre-specified according to the used storage space. For confidentiality, the owner encrypts the data before sending to cloud servers. After data outsourcing, the owner can interact with the CSP to perform block-level operations on the file. These operations

includes modify, insert, append, and delete specific blocks. In addition, the owner enforces access control by granting or revoking access rights to the outsourced data.

An authorized user sends a data-access request to the CSP, and receives the data file in an encrypted form that can be decrypted using a secret key generated by the authorized user (more details will be explained later). We assume that the interaction between the owner and the authorized users to authenticate their identities has already been completed, and it is not considered in this work. Moreover, all authorized users have the same privileges, i.e., the issue of access hierarchy is outside the current scope.

The TTP is an independent entity, and thus has no incentive to collude with any party in the system. However, any possible leakage of data towards the TTP must be prevented to keep the outsourced data private. The TTP and the CSP are always online, while the owner is *intermittently* online. The authorized users are able to access the data file from the CSP even when the owner is offline. Throughout this paper, the terms cloud server and cloud service provider are used interchangeably.

Threat model. The CSP is untrusted, and thus the confidentiality and integrity of data in the cloud may be at risk. For economic incentives and maintaining a reputation, the CSP may 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. To save the computational resources, the CSP may totally ignore the data-update requests issued by the owner, or execute just a few of them. Hence, the CSP may return damaged or stale data for any access request from the authorized users. Furthermore, the CSP may not honor the access rights created by the owner, and permit unauthorized access for misuse of confidential data.

On the other hand, a data owner and authorized users may collude and falsely accuse the CSP to get a certain amount of reimbursement. They may dishonestly claim that data integrity over cloud servers has been violated, or the CSP has returned a stale file that does not match the most recent modifications issued by the owner.

Security requirements. *Confidentiality*: outsourced data must be protected from the TTP, the CSP, and users that are not granted access. *Integrity*: outsourced data is required to remain intact on cloud servers. The data owner and authorized users must be enabled to recognize data corruption over the CSP side. *Newness*: receiving the most recent version of the outsourced data file is an imperative requirement of cloud-based storage systems. There must be a detection mechanism if the CSP ignores any data-update requests issued by the owner. *Access control*: only authorized users are allowed to access the outsourced data. Revoked users can read unmodified data, however, they must not be able to read updated/new blocks. *CSP's defence*: the CSP must be safeguarded against false accusations that may be claimed by dishonest owner/users, and such a malicious behavior is required to be revealed.

Combining the confidentiality, integrity, newness, access control, and CSP's defence properties in the proposed scheme enables the mutual trust between the data owner and the CSP. Thus, the owner can

benefit from the wide range of facilities offered by the CSP, and at the same time, the CSP can mitigate the concern of cheating customers.

4 System Preliminaries

4.1 Lazy Revocation

The proposed scheme in this work allows the data owner to revoke the right of some users for accessing the outsourced data. In lazy revocation, it is acceptable for revoked users to read (decrypt) *unmodified* data blocks. However, updated or new blocks must not be accessed by such revoked users. The notation of lazy revocation was first introduced in [41]. The idea is that allowing revoked users to read unchanged data blocks is not a significant loss in security. This is equivalent to accessing the blocks from cashed copies. Updated or new blocks following a revocation are encrypted under new keys. Lazy revocation trades re-encryption and data access cost for a degree of security. However, it causes fragmentation of encryption keys, i.e., data blocks could have more than one key. Lazy revocation has been incorporated into many cryptographic systems [35], [39], [42], [43].

4.2 Key Rotation

Key rotation [26] is a technique in which a sequence of keys can be generated from an initial key and a master secret key. The sequence of keys has two main properties: (i) only the owner of the master secret key is able to generate the next key in the sequence from the current key, and (ii) any authorized user knowing a key in the sequence is able to generate all previous versions of that key. In other words, given the i-th key K_i in the sequence, it is computationally infeasible to compute keys $\{K_l\}$ for $l > i$ without having the master secret key, but it is easy to compute keys $\{K_j\}$ for $j < i$.

The first property enables the data owner to revoke access to the data by producing new keys in the sequence, which are used to encrypt updated/new blocks following a revocation (lazy revocation). It is intended to prevent a user revoked during the i-th time from getting access to data blocks encrypted during the l-th time for $l > i$.

The second property allows authorized users to maintain access to blocks that are encrypted under older versions of the current key. It enables the data owner to transfer only a single key K_i to authorized users for accessing all data blocks that are encrypted under keys $\{K_j\}_{j \leq i}$ (rather than transferring a potentially large set of keys $\{K_1, K_2, \ldots, K_i\}$). Thus, the second property reduces the communication overhead on the owner side.

The proposed scheme in this work utilizes the key rotation technique [26]. Let $N = pq$ denote the RSA modulus ($p \,\&\, q$ are prime numbers), a public key = (N, e), and a master secret key d. The key d is known only to the data owner, and $ed \equiv 1 \bmod (p-1)(q-1)$.

Whenever a user's access is revoked, the data owner generates a new key in the sequence (*rotating forward*). Let ctr indicate the index/version number of the current key in the keys sequence. The owner

generates the next key by exponentiating K_{ctr} with the master secret key d: $K_{ctr+1} = K_{ctr}^{d} \bmod N$. Authorized users can recursively generate older versions of the current key by exponentiating with the public key component e: $K_{ctr-1} = K_{ctr}^{e} \bmod N$ (*rotating backward*). The RSA encryption is used as a pseudorandom number generator; it is unlikely that repeated encryption results in cycling, for otherwise, it can be used to factor the RSA modulus N [44].

4.3 Broadcast Encryption

Broadcast encryption (bENC) [45], [46] allows a broadcaster to encrypt a message for an arbitrary subset of a group of users. The users in the subset are only allowed to decrypt the message. However, even if all users outside the subset collude they cannot access the encrypted message. Such systems have the collusion resistance property, and are used in many practical applications including TV subscription services and DVD content protection. The proposed scheme in this work uses bENC [45] to enforce access control in outsourced data. The bENC [45] is composed of three algorithms: SETUP, ENCRYPT, and DECRYPT.

SETUP. This algorithm takes as input the number of system users n. It defines a bilinear group $\mathbb{G}$ of prime order p with a generator g, a cyclic multiplicative group $\mathbb{G}_T$, and a bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, which has the properties of bilinearity, computability, and non-degeneracy [47]. The algorithm picks a random $\alpha \in \mathbb{Z}_p$, computes $g_i = g^{(\alpha^i)} \in \mathbb{G}$ for $i = 1, 2, \ldots, n, n+2, \ldots, 2n$, and sets $v = g^{\gamma} \in \mathbb{G}$ for $\gamma \in_R \mathbb{Z}_p$. The outputs are a public key $PK = (g, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}, v) \in \mathbb{G}^{2n+1}$, and n private keys $\{d_i\}_{1 \leq i \leq n}$, where $d_i = g_i^{\gamma} \in \mathbb{G}$.

ENCRYPT. This algorithm takes as input a subset $S \subseteq \{1, 2, \ldots, n\}$, and a public key PK. It outputs a pair (Hdr, K), where Hdr is called the header (broadcast ciphertext), and K is a message encryption key. Hdr = $(C_0, C_1) \in \mathbb{G}^2$, where for $t \in_R \mathbb{Z}_p, C_0 = g^t$ and $C_1 = (v \cdot \prod_{j \in S} g_{n+1-j})^t$. The key $K = \hat{e}(g_{n+1}, g)^t$ is used to encrypt a message M (symmetric encryption) to be broadcast to the subset S.

DECRYPT. This algorithm takes as input a subset $S \subseteq \{1, 2, \ldots, n\}$, a user-ID $i \in \{1, 2, \ldots, n\}$, the private key d_i for user i, the header Hdr = (C_0, C_1), and the public key PK. If $i \in S$, the algorithm outputs the key $K = \hat{e}(g_i, C_1) / \hat{e}(d_i \cdot \prod_{\substack{j \in S \\ j \neq i}} g_{n+1-j+i}, C_0)$, which can be used to decrypt the encrypted version of M.

In the above construction of the bENC [45], a private key contains only one element of $\mathbb{G}$, and the broadcast ciphertext (Hdr) consists of two elements of $\mathbb{G}$. On the other hand, the public key PK is comprised of $2n + 1$ elements of $\mathbb{G}$. A second construction, which is a generalization of the first one was presented in [45] to trade the PK size for the Hdr size. The main idea is to run multiple parallel instances of the first construction, where each instance can broadcast to at most B users. Setting $B = \lfloor \sqrt{n} \rfloor$ results in a system with $O(\sqrt{n})$ elements of $\mathbb{G}$ for each of PK and Hdr. The private key is still just one element.

In this work, we utilize the second construction to achieve a balance between the sizes of PK and Hdr. For an organization (data owner) with 10^5 users, each of PK and Hdr contains only 317 elements of $\mathbb{G}$.

5 Proposed Cloud-Based Storage Scheme

5.1 Warmup Discussion

Before presenting our main scheme, we discuss a straightforward solution. Once the data has been outsourced to a remote CSP, which may not be trustworthy, the owner loses the direct control over the sensitive data. This lack of control raises the data owner's concerns about the integrity of data stored in the cloud. Conversely, a dishonest owner may falsely claim that the data stored in the cloud is corrupted to get some compensation. This mutual distrust between the data owner and the CSP, if not properly handled, may hinder the successful deployment of cloud architecture.

A straightforward solution to detect cheating from any side is through using authentication tags (digital signatures). For a file $F = \{b_j\}_{1 \leq j \leq m}$, the owner attaches a tag $\texttt{OWN}\sigma_j$ with each block before outsourcing. The tags are generated per block not per file to enable dynamic operations at the block level without retrieving the whole outsourced file. The owner sends $\{b_j, \texttt{OWN}\sigma_j\}_{1 \leq j \leq m}$ to the CSP, where the tags $\{\texttt{OWN}\sigma_j\}_{1 \leq j \leq m}$ are first verified. In case of failed verification, the CSP rejects to store the data blocks and asks the owner to re-send the correct tags. If the tags are valid, both the blocks and the tags are stored on the cloud servers. The tags $\{\texttt{OWN}\sigma_j\}_{1 \leq j \leq m}$ achieve non-repudiation from the owner side. When an authorized user (or the owner) requests to retrieve the data file, the CSP sends $\{b_j, \texttt{OWN}\sigma_j, \texttt{CSP}\sigma_j\}_{1 \leq j \leq m}$, where $\texttt{CSP}\sigma_j$ is the CSP's signature/tag on $b_j||\texttt{OWN}\sigma_j$. The authorized user first verifies the tags $\{\texttt{CSP}\sigma_j\}_{1 \leq j \leq m}$. In case of failed verification, the user asks the CSP to re-perform the transmission process. If $\{\texttt{CSP}\sigma_j\}_{1 \leq j \leq m}$ are valid tags, the user then verifies the owner's tag $\texttt{OWN}\sigma_j$ on the block $b_j \; \forall \; j$. If any tag $\texttt{OWN}\sigma_j$ is not verified, this indicates the corruption of data over the cloud servers. The CSP cannot repudiate such corruption for the owner's tags $\{\texttt{OWN}\sigma_j\}_{1 \leq j \leq m}$ are previously verified and stored by the CSP along with the data blocks. Since the CSP's signatures $\{\texttt{CSP}\sigma_j\}_{1 \leq j \leq m}$ are attached with the received data, a dishonest owner cannot falsely accuse the CSP regarding data integrity.

Although the previous straightforward solution can detect cheating from either side, it cannot guarantee the newness property of the outsourced data; the CSP can replace the new blocks and tags with old versions without being detected (*replay attack*). The above solution increases the storage overhead – especially for large files in order of gigabytes – on the cloud servers as each outsourced block is attached with a tag. Moreover, there is an increased computation overhead on different system components; the data owner generates a signature for each block, the CSP performs a signature verification for each outsourced block, and the authorized user (or the owner) verifies two signatures for each received block from the cloud servers. Thus, for a file F containing m blocks, the straightforward solution requires $2m$ signature generations and $3m$ signature verifications, which may be computationally a challenging task for large data files. For example, if the outsourced file is of size 1GB with 4KB block size, the straightforward solution requires 2^{19} signature generations and 3×2^{18} signature verifications.

If the CSP receives the data blocks from a trusted entity (other than the owner), the block tags and

the signature operations are not needed since the trusted entity has no incentive for repudiation or collusion. Therefore, delegating a small part of the owner's work to the TTP reduces both the storage and computation overheads. However, the outsourced data must be kept private and any possible leakage of data towards the TTP must be prevented.

5.2 Overview and Rationale

The proposed scheme in this work addresses important issues related to outsourcing data storage: dynamic data, newness, mutual trust, and access control. The owner is allowed to update and scale the outsourced data file. Validating such dynamic data and its newness property requires the knowledge of some metadata that reflects the most recent modifications issued by the owner. Moreover, it requires the awareness of block indices to guarantee that the CSP has inserted, added, or deleted the blocks at the requested positions. To this end, the proposed scheme is based on using *combined* hash values and a small data structure, which we call block status table (BST). The TTP establishes the mutual trust among different system components in an indirect way.

For enforcing access control of the outsourced data, the proposed scheme utilizes and combines three cryptographic techniques: bENC, lazy revocation, and key rotation. The bENC enables a data owner to encrypt some secret information to only authorized users allowing them to access the outsourced data file. Through lazy revocation, revoked users can read unmodified data blocks, while updated/new blocks are encrypted under new keys generated from the secret information broadcast to the authorized users. Using key rotation, the authorized users are able to access both updated/new blocks and unmodified ones that are encrypted under older versions of the current key.

5.3 Notations

- F is a data file to be outsourced, and is composed of a sequence of m blocks, i.e., $F = \{b_1, b_2, \ldots, b_m\}$
- h is a cryptographic hash function
- DEK is a data encryption key
- E_{DEK} is a symmetric encryption algorithm under DEK, e.g., AES (advanced encryption standard)
- E^{-1}_{DEK} is a symmetric decryption algorithm under DEK
- $\widetilde{F}$ is an encrypted version of the file blocks
- FH_{TTP} is a combined hash value for $\widetilde{F}$, and is computed and stored by the TTP
- TH_{TTP} is a combined hash value for the BST, and is computed and stored by the TTP
- ctr is a counter kept by the data owner to indicate the version of the most recent key
- $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$ is a rotator, where $\mathsf{bENC}(K_{ctr})$ is a broadcast encryption of the key K_{ctr}
- $\oplus$ is an XOR operator

5.4 Block Status Table

The block status table (BST) is a small *dynamic* data structure used to reconstruct and access file blocks outsourced to the CSP. The BST consists of three columns: serial number ($\mathcal{SN}$), block number ($\mathcal{BN}$), and key version ($\mathcal{KV}$). $\mathcal{SN}$ is an indexing to the file blocks. It indicates the physical position of each block in the data file. $\mathcal{BN}$ 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 $\mathcal{SN}$. $\mathcal{KV}$ indicates the version of the key that is used to encrypt each block in the data file.

The BST is implemented as a linked list to simplify the insertion and deletion of table entries. During implementation, $\mathcal{SN}$ is not needed to be stored in the table; $\mathcal{SN}$ is considered to be the entry/table index. Thus, each table entry contains just two integers $\mathcal{BN}$ and $\mathcal{KV}$ (8 bytes), i.e., the total table size is $8m$ bytes, where m is the number of file blocks.

When a data file is initially created, the owner initializes both ctr and $\mathcal{KV}$ of each block to 1. If block modification or insertion operations are to be performed following a revocation, ctr is incremented by 1 and $\mathcal{KV}$ of that modified/new block is set to be equal to ctr.

Fig. 2 shows some examples demonstrating the changes in the BST due to dynamic operations on a data file $F = \{b_j\}_{1 \leq j \leq 8}$. When the file blocks are initially created (Fig. 2a), ctr is initialized to 1, $\mathcal{SN}_j$ = $\mathcal{BN}_j$ = j, and $\mathcal{KV}_j$ = 1: $1 \leq j \leq 8$. Fig. 2b shows no change for updating the block at position 5 since no revocation is performed. To insert a new block after position 3 in the file F, Fig. 2c shows that a new entry $\langle 4, 9, 1 \rangle$ is inserted in the BST 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 key used for encryption.

A first revocation in the system increments ctr by 1 (ctr = 2). Modifying the block at position 5 following a revocation (Fig. 2d) results in setting $\mathcal{KV}_5$ = ctr. Thus, the table entry at position 5 becomes $\langle 5, 4, 2 \rangle$. Fig. 2e shows that a new block is to be inserted after position 6 following a second revocation, which increments ctr to be 3. In Fig. 2e, a new table entry $\langle 7, 10, 3 \rangle$ is inserted after $\mathcal{SN}_6$, where $\mathcal{KV}_7$ is set to be equal to ctr (the most recent key version). Deleting a block at position 2 from the data file requires deleting the table entry at $\mathcal{SN}_2$ and shifting all subsequent entries one position up (Fig. 2f). Note that during all dynamic operations, $\mathcal{SN}$ indicates the actual physical positions of the data blocks in F.

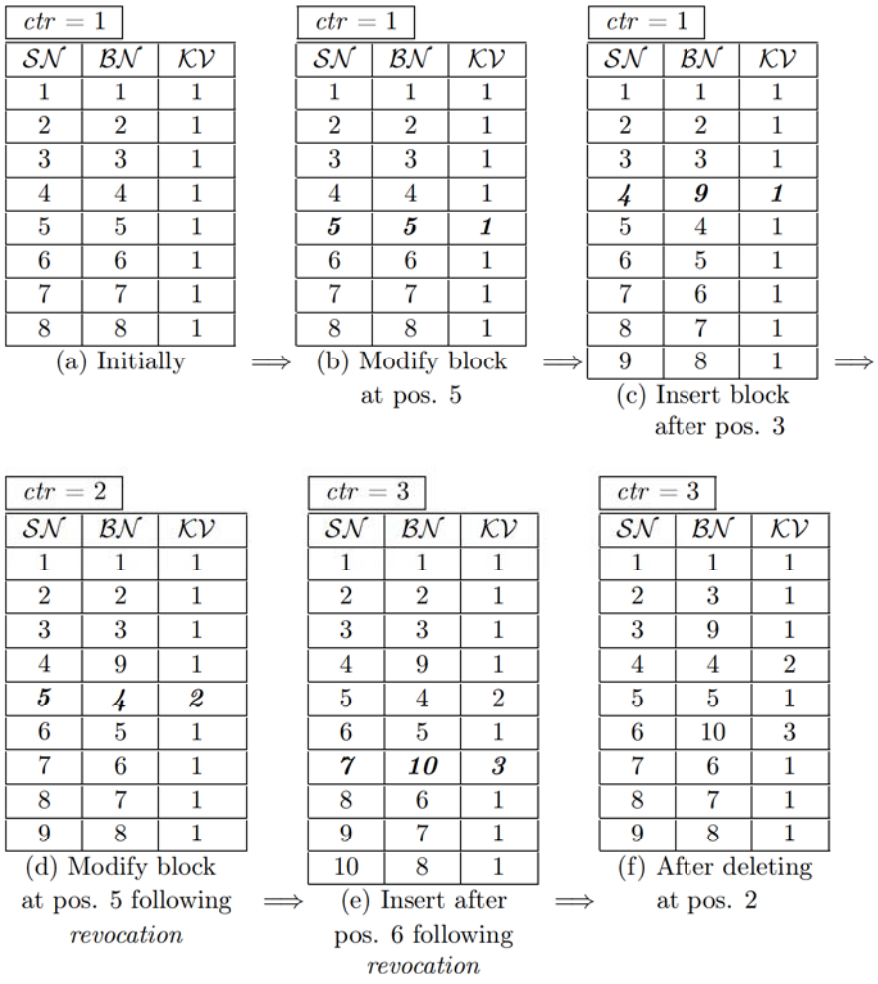

Fig. 2: Changes in the BST due to different dynamic operations on a file $F = \{b_j\}_{1 \le j \le 8}$. $\mathcal{SN}$ is the serial number, $\mathcal{BN}$ is the block number, and $\mathcal{KV}$ is the key version.

5.5 Procedural Steps of the Proposed Scheme

- ***Setup and File Preparation***. The setup is done only once during the life time of the data storage system, which may be for tens of years. The system setup has two parts: one is done on the owner side, and the other is done on the TTP side.
  - **Owner Role**. The data owner initializes ctr to 1, and generates an initial secret key K_{ctr}/K_1. K_{ctr} can be rotated forward following user revocations, and rotated backward to enable authorized users to access blocks that are encrypted under older versions of K_{ctr}.

    For a file $F = \{b_j\}_{1 \le j \le m}$, the owner generates a BST with $\mathcal{SN}_j = \mathcal{BN}_j = j$ and $\mathcal{KV}_j = ctr$. To achieve privacy-preserving, the owner creates an encrypted file version $\widetilde{F} = \{\tilde{b}_j\}_{1 \le j \le m}$,

where $\tilde{b}_j = E_{DEK}(\mathcal{BN}_j||b_j)$ and $DEK = h(K_{ctr})$.[2] Moreover, the owner creates a rotator $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$, where bENC enables only authorized users to decrypt K_{ctr} and access the outsourced file. The owner sends $\{\widetilde{F}, \text{BST}, Rot\}$ to the TTP, and deletes the data file from its local storage.

Embedding $\mathcal{BN}_j$ with the block b_j during the encryption process helps in reconstructing the file blocks in the correct order. If the encrypted blocks $\{\tilde{b}_j\}_{1\leq j\leq m}$ are not corrupted over cloud servers, but randomly delivered to an authorized user, the latter can utilize the embedded $\mathcal{BN}_j$ and the BST to orderly reconstruct the data file F. More details will be explained later.

- **TTP Role**. As previously explained, a small part of the owner's work is delegated to the TTP to reduce the storage overhead and lower the overall system computation. For the TTP to resolve disputes that may arise regarding data integrity/newness, it computes and locally stores combined hash values for the encrypted file $\widetilde{F}$ and the BST. The TTP computes $FH_{TTP} = \oplus_{j=1}^{m} h(\tilde{b}_j)$ and $TH_{TTP} = \oplus_{j=1}^{m} h(\mathcal{BN}_j||\mathcal{KV}_j)$, then sends $\{\widetilde{F}, \text{BST}\}$ to the CSP. The TTP keeps only FH_{TTP} and TH_{TTP} on its local storage.

Remark 2. The BST is used by the authorized users to reconstruct and access the outsourced data file. The proposed scheme in this work assumes that the data owner is *intermittently* online and the authorized users are enabled to access the data file even when the owner is offline. To this end, the CSP stores a copy of the BST along with the outsourced data file. When an authorized user requests to access the data, the CSP responds by sending both the BST and the encrypted file $\widetilde{F}$.

Moreover, the BST is used during each dynamic operation on the outsourced data file, where one table entry is modified/inserted/deleted with each dynamic change on the block level. If the BST is stored only on the CSP side, it needs to be retrieved and validated each time the data owner wants to issue a dynamic request on the outsourced file. To avoid such communication and computation overheads, the owner keeps a local copy of the BST, and thus there are two copies of the BST: one is stored on the owner side referred to as BST_O, and the other is stored on the CSP side referred to as BST_C. Recall that the BST is a small dynamic data structure with a table entry size = 8 bytes. For 1GB file with 4KB block size, the BST size is only 2MB (0.2% of the file size). Table 1 summarizes the data stored by each component in the proposed scheme.

TABLE 1: Data stored by each component in the proposed scheme.

Owner	TTP	CSP
ctr, K_{ctr}, BST_O	Rot, FH_{TTP}, TH_{TTP}	$\widetilde{F}$, BST_C

2. Hash in needed to compress the size of K_{ctr}

■ ***Dynamic Operations on the Outsourced Data***. The dynamic operations in the proposed scheme are performed at the block level via a request in the general form $\langle \mathsf{BlockOp},\ \mathrm{TEntry}_{\mathsf{BlockOp}},\ j,\ \mathcal{KV}_j,\ h(\tilde{b}_j),\ \texttt{RevFlag},\ b^* \rangle$, where BlockOp corresponds to block modification (denoted by BM), block insertion (denoted by BI), or block deletion (denoted by BD). $\mathrm{TEntry}_{\mathsf{BlockOp}}$ indicates an entry in BST_O corresponding to the issued dynamic request. The parameter j indicates the block index on which the dynamic operation is to be performed, $\mathcal{KV}_j$ is the value of the key version at index j of BST_O before running a modification operation, and $h(\tilde{b}_j)$ is the hash value of the block at index j before modification/deletion. `RevFlag` is a 1-bit flag (true/false and is initialized to false) to indicate whether a revocation has been performed, and b^* is the new block value.

- **Modification**. Data modification is one of the most frequently used dynamic operations in the outsourced data. For a file $F = \{b_1, b_2, \ldots, b_m\}$, suppose the owner wants to modify a block b_j with a block b'_j. Fig. 3 describes the steps performed by each system component (owner, CSP, and TTP) during block modification. The owner uses the technique of one-sender-multiple-receiver (OSMR) transmission to send the modify request to both the CSP and the TTP.

  The TTP updates the combined hash value FH_{TTP} for $\widetilde{F}$ through the step $FH_{TTP} = FH_{TTP} \oplus h(\tilde{b}_j) \oplus h(\tilde{b}'_j)$, which simultaneously replaces the hash of the old block $h(\tilde{b}_j)$ with the new one $h(\tilde{b}'_j)$. This is possible due to the basic properties of the $\oplus$ operator. The same idea is used when `RevFlag` = true to update the combined hash value TH_{TTP} on the TTP side by replacing the hash of the old table entry at index j with the hash of the new value.
- **Insertion**. In a block insertion operation, the owner wants to insert a new block $\bar{b}$ *after* index j in a file $F = \{b_1, b_2, \ldots, b_m\}$, i.e., the newly constructed file $F' = \{b_1, b_2, \ldots, b_j, \bar{b}, \ldots, b_{m+1}\}$, where $b_{j+1} = \bar{b}$. The block insertion operation changes the logical structure of the file, while block modification does not. Fig. 4 describes the steps performed by each system component (owner, CSP, and TTP) during block insertion.
- **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**. Block deletion operation is the opposite of the insertion operation. When one block is deleted all subsequent blocks are moved one step forward. Fig. 5 describes the steps performed by each system component (owner, CSP, and TTP) during block deletion. The step $FH_{TTP} = FH_{TTP} \oplus h(\tilde{b}_j)$ is used to delete the hash value of the block $\tilde{b}_j$ from the combined hash FH_{TTP} (properties of $\oplus$ operator). The same idea is used with the TH_{TTP} value.

■ ***Data Access and Cheating Detection***. Fig. 6 shows the verifications performed for the data received from the CSP, and presents how authorized users get access to the outsourced file.

An authorized user sends a data-access request to both the CSP and the TTP to access the outsourced file. For achieving non-repudiation, the CSP generates two signatures σ_F and σ_T for $\widetilde{F}$ and BST_C,

/* *Modification of a block* b_j *with* b'_j *for the outsourced file* */

/* `RevFlag` *is initialized to false* */

Data Owner

1) **If** the access of one or more users has been revoked **then**
   a) Rolls K_{ctr} forward (using key rotation)
   b) Increments ctr = ctr + 1, and sets `RevFlag` = true
   c) Copies $\mathcal{KV}_j$ from BST_O to $\overline{\mathcal{KV}}_j$ (i.e., $\overline{\mathcal{KV}}_j = \mathcal{KV}_j$)
   d) Sets $\mathcal{KV}_j = ctr$ in BST_O, and generates $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$
   e) Sends Rot to the TTP
2) Creates an encrypted block $\tilde{b}'_j = E_{DEK}(\mathcal{BN}_j||b'_j)$, where $DEK = h(K_{ctr})$
3) Forms a block-modify table entry $\text{TEntry}_{\mathsf{BM}} = \{\mathcal{BN}_j, \mathcal{KV}_j\}$
4) Sends a modify request $\langle \mathsf{BM},\ \text{TEntry}_{\mathsf{BM}},\ j,\ \overline{\mathcal{KV}}_j,\ h(\tilde{b}_j),\ \texttt{RevFlag},\ \tilde{b}'_j\rangle$ to both the CSP and the TTP (OSMR transmission), where $h(\tilde{b}_j)$ is the hash of the outsourced block to be modified. The $\overline{\mathcal{KV}}_j$ is not sent in the modify request if `RevFlag` = false
5) The CSP accepts the modify request only if $\{\mathcal{BN}_j, \overline{\mathcal{KV}}_j\}$ sent from the owner matches $\{\mathcal{BN}_j, \mathcal{KV}_j\}$ in BST_C, and $h(\tilde{b}_j)$ is equal to the hash of the block $\tilde{b}_j$ on the cloud server (to guarantee that correct values are sent to the TTP)

CSP /* *upon accepting the modify request from the owner* */

1) Replaces the block b_j with b'_j in the outsourced file $\widetilde{F}$
2) **If** `RevFlag` = true **then**
   Updates the table entry at index j of BST_C using $\text{TEntry}_{\mathsf{BM}}$ components

TTP

1) Updates $FH_{TTP} = FH_{TTP} \oplus h(\tilde{b}_j) \oplus h(\tilde{b}'_j)$
2) **If** `RevFlag` = true **then**
   a) Updates the previously stored Rot with the newly received value
   b) Updates $TH_{TTP} = TH_{TTP} \oplus h(\mathcal{BN}_j||\overline{\mathcal{KV}}_j) \oplus h(\mathcal{BN}_j||\mathcal{KV}_j)$

Fig. 3: Block modification procedure in the proposed scheme.

respectively. The authorized user receives $\{\widetilde{F}$, BST_C σ_F, $\sigma_T\}$ from the CSP, and $\{FH_{TTP}$, TH_{TTP}, $Rot\}$ from the TTP. The authorized user verifies the signatures, and proceeds with the data access procedure only if both signatures are valid.

The authorized user verifies the contents of BST_C entries by computing a combined hash value $TH_U = \oplus_{j=1}^{m} h(\mathcal{BN}_j||\mathcal{KV}_j)$, and comparing it with the authentic value TH_{TTP} received from the

/ Insertion of a block $\bar{b}$ after index j in the outsourced file */*

/ `RevFlag` is initialized to false */*

Data Owner

1) **If** the access of one or more users has been revoked **then**
   a) Rolls K_{ctr} forward (using key rotation)
   b) Increments ctr = ctr + 1, and sets `RevFlag` = true
   c) Generates $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$
   d) Sends Rot to the TTP
2) Constructs a new block-insert table entry $\text{TEntry}_{\mathsf{BI}} = \{\mathcal{BN}_{j+1}, \mathcal{KV}_{j+1}\} = \{1 + Max\{\mathcal{BN}_j\}_{1\leq j\leq m}, ctr\}$, and inserts this entry in BST_O after index j
3) Creates an encrypted block $\tilde{\bar{b}} = E_{DEK}(\mathcal{BN}_j||\bar{b})$, where $DEK = h(K_{ctr})$
4) Sends an insert request $\langle \mathsf{BI},\ \text{TEntry}_{\mathsf{BI}},\ j,\ null,\ null,\ \texttt{RevFlag},\ \tilde{\bar{b}}\rangle$ to both the CSP and the TTP (OSMR transmission)

CSP */* upon receiving the insert request from the owner */*

1) Inserts the block $\tilde{\bar{b}}$ after index j in the outsourced file $\widetilde{F}$
2) Inserts the table entry $\text{TEntry}_{\mathsf{BI}}$ after index j in the BST_C

TTP

1) Updates $FH_{TTP} = FH_{TTP} \oplus h(\tilde{\bar{b}})$
2) Updates $TH_{TTP} = TH_{TTP} \oplus h(\mathcal{BN}_{j+1}||\mathcal{KV}_{j+1})$
3) **If** `RevFlag` = true **then**
   Replaces the previously stored Rot with the newly received value

Fig. 4: Block insertion procedure in the proposed scheme.

TTP. If the authorized user claims that $TH_U \neq TH_{TTP}$, a report is issued to the owner and the TTP is invoked to determine the dishonest party.

In case of $TH_U = TH_{TTP}$, the authorized user continues to verify the contents of the file $\widetilde{F}$. A combined hash value $FH_U = \oplus_{j=1}^{m} h(\tilde{b}_j)$ is computed and compared with FH_{TTP}. If there is a dispute that $FH_U \neq FH_{TTP}$, the owner is informed and we resort to the TTP to resolve such a conflict.

For the authorized user to access the encrypted file $\widetilde{F} = \{\tilde{b}_j\}_{1\leq j\leq m}$, BST_C and Rot are used to generate the key DEK that decrypts the block $\tilde{b}_j$. $\mathsf{bENC}(K_{ctr})$ is decrypted to get the most recent key K_{ctr}. Using the key rotation technique, the authorized user rotates K_{ctr} backward with each block until it reaches the version that is used to decrypt the block $\tilde{b}_j$. Both ctr and the key version

/ Deletion of a block b_j from the outsourced file */*

Data Owner

1) Copies the entry at index j from BST_O to a block-delete table entry $\text{TEntry}_{\mathsf{BD}} = \{\mathcal{BN}_j, \mathcal{KV}_j\}$
2) Deletes the entry at index j from BST_O
3) Sends a delete request $\langle \mathsf{BD},\ \text{TEntry}_{\mathsf{BD}},\ j,\ null,\ h(\tilde{b}_j),\ \text{false},\ null \rangle$ to both the CSP and the TTP (OSMR transmission), where $h(\tilde{b}_j)$ is the hash of the outsourced block to be deleted
4) The CSP accepts the delete request only if $\text{TEntry}_{\mathsf{BD}}$ sent from the owner matches $\{\mathcal{BN}_j, \mathcal{KV}_j\}$ in BST_C and $h(\tilde{b}_j)$ is equal to the hash of the block $\tilde{b}_j$ on the cloud server (to guarantee that correct values are sent to the TTP).

CSP */* upon receiving the delete request from the owner */*

1) Deletes the block at index j (block $\tilde{b}_j$) from the outsourced file $\widetilde{F}$
2) Deletes the entry at index j from the BST_C

TTP

1) Updates $FH_{TTP} = FH_{TTP} \oplus h(\tilde{b}_j)$
2) Updates $TH_{TTP} = TH_{TTP} \oplus h(\mathcal{BN}_j || \mathcal{KV}_j)$

Fig. 5: Block deletion procedure in the proposed scheme.

$\mathcal{KV}_j$ can determine how many rotation steps for K_{ctr} with each block $\tilde{b}_j$. Decrypting the block $\tilde{b}_j$ returns $(\mathcal{BN}_j || b_j)$. $\mathcal{BN}_j$ and BST_C are utilized to get the physical block position $\mathcal{SN}_j$ into which the block b_j is inserted, and thus the file F is reconstructed in plain form.

Optimization. In Fig. 6, the backward key rotation done in the *inner* for loop of step 7.b can be highly optimized by computing a set of keys $Q = \{K_i\}$ from K_{ctr}. Each key K_i in Q is the result of rotating K_{ctr} backward $ctr - i$ times. For example, if ctr = 20, a set $Q = \{K_1, K_5, K_{10}, K_{15}\}$ can be computed from K_{ctr}. To decrypt a block $\tilde{b}_j$, the authorized user chooses one key K_i from Q, which has the minimum *positive* distance $i - \mathcal{KV}_j$. The key K_i is then rotated backward to get the actual key that is used to decrypt the block $\tilde{b}_j$. A relatively large portion of the outsourced data is kept unchanged on the CSP, and thus K_1 from Q can be used to decrypt many blocks without any further key rotation. The size of the set Q is negligible compared with the size of the received data file.

Fig. 7 shows how the TTP determines the dishonest party in the system. The TTP verifies the signatures σ_T and σ_F, which are previously verified and accepted by the authorized user. If any signature is invalid, this indicates that the owner/user is dishonest for corrupting either the data or the signatures. In case of valid signatures, the TTP computes temporary combined hash values $TH_{temp} = \oplus_{j=1}^{m} h(\mathcal{BN}_j || \mathcal{KV}_j)$ and $FH_{temp} = \oplus_{j=1}^{m} h(\tilde{b}_j)$. If $TH_{temp} \neq TH_{TTP}$ or $FH_{temp} \neq FH_{TTP}$,

1) An authorized user sends a data-access request to both the CSP and the TTP
2) The CSP responds by sending the outsourced file $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$ associated with a signature σ_F (CSP's signature on the entire file), and sending BST_C associated with a signature σ_T (CSP's signature on the entire table) to the authorized user
3) The authorized user verifies σ_F and σ_T, and accepts the data only if σ_F and σ_T are valid signatures
4) The TTP sends FH_{TTP} , TH_{TTP}, and $Rot = \langle ctr, \mathsf{bENC}(K_{ctr}) \rangle$ to the authorized user
5) **Verification of the BST_C entries**
   a) The authorized user computes $TH_U = \oplus_{j=1}^{m} h(\mathcal{BN}_j || \mathcal{KV}_j)$
   b) **If** the authorized user claims that $TH_U \neq TH_{TTP}$ **then** report "integrity violation" to the owner and invoke cheating detection procedure (Fig. 7)
6) **Verification of the data file $\widetilde{F}$**
   a) The authorized user computes $FH_U = \oplus_{j=1}^{m} h(\tilde{b}_j)$
   b) **If** the authorized user claims that $FH_U \neq FH_{TTP}$ **then** report "integrity violation" to the owner and invoke cheating detection procedure (Fig. 7)
7) **Data access**
   a) The authorized user gets K_{ctr} by decrypting $\mathsf{bENC}(K_{ctr})$ part in Rot
   b) **for** j = 1 to m **do**

   / rotate backward the current K_{ctr} to the version that is used to decrypt the block $\tilde{b}_j$ */*

   – Set $K_j = K_{ctr}$
   – **for** i = 1 to ctr - $\mathcal{KV}_j$ **do**

   $K_j = (K_j)^e \bmod N$ */* N is the RSA modulus* and (N, e) *is the RSA public key */*

   end for
   – $(\mathcal{BN}_j || b_j) = E_{DEK}^{-1}(\tilde{b}_j)$, where $DEK = h(K_j)$
   – Get the physical position $\mathcal{SN}_j$ of b_j using $\mathcal{BN}_j$ and BST_C
   – The authorized user places b_j in the correct order of the decrypted file F

   end for

Fig. 6: Data access procedure in the proposed scheme.

this indicates that the CSP is dishonest for sending corrupted data to the authorized user, otherwise the owner/user is dishonest for falsely claiming integrity violation of received data.

Cheating Detection Procedure: determination of the dishonest party.

The TTP is invoked to determine which component is misbehaving as follows.

1) The TTP verifies σ_T and σ_F
2) **If** any signature verification fails **then**
   TTP reports "dishonest owner/user" and exits
3) The TTP computes $TH_{temp} = \oplus_{j=1}^{m} h(\mathcal{BN}_j||\mathcal{KV}_j)$ and $FH_{temp} = \oplus_{j=1}^{m} h(\tilde{b}_j)$
4) **If** $TH_{temp} \neq TH_{TTP}$ **or** $FH_{temp} \neq FH_{TTP}$ **then**
   TTP reports "dishonest CSP" and exits /* *data is corrupted* */
   else
   TTP reports "dishonest owner/user" and exits /* *data is NOT corrupted* */

Fig. 7: Cheating detection procedure in the proposed scheme.

6 Security Analysis

In this section we investigate the security of the proposed scheme by analyzing its fulfillment of the security requirements described in Section 3, namely, confidentiality, integrity, newness, access control, and CSP's defence.

Data confidentiality. We need to prove that the CSP, the TTP, and unauthorized users cannot access the outsourced data.

Theorem 1. *The proposed scheme preserves the confidentiality of the outsourced data against the CSP, the TTP, and unauthorized users*.

Proof (Sketch). Before outsourcing a data file $F = \{b_j\}_{1 \leq j \leq m}$, the owner generates an encrypted version $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$, where $\tilde{b}_j = E_{DEK}(\mathcal{BN}_j||b_j)$, and $DEK = h(K_{ctr})$. The encrypted data file $\widetilde{F}$ is sent to the TTP for computing FH_{TTP} and to the CSP for storage.

Based on the security of the underlying symmetric encryption algorithm E_{DEK}, the confidentiality of the outsourced file $\widetilde{F}$ is preserved on the CSP side (e.g., our proposed scheme utilizes AES – a standardized encryption algorithm by NIST [48] – with 128-bit security level to achieve a robust security requirement).

The data confidentiality on the TTP side is based on the security of the underlying broadcast encryption algorithm bENC. To decrypt $\widetilde{F}$, the key K_{ctr} is needed to generate DEK. The TTP stores $Rot = \langle ctr, \text{bENC}(K_{ctr})\rangle$. Thus, for the TTP to get K_{ctr}, bENC must be broken (the proposed scheme utilizes bENC [45], which is proved to be semantically secure).

bENC also prevents unauthorized users from getting K_{ctr} to access the data file $\widetilde{F}$. Moreover, based on the hardness of the RSA problem [49], revoked users who possess K_l ($l < ctr$) are not able to generate

K_{ctr}. Hence, such revoked users can access only stale data blocks, while updated or new blocks encrypted using K_{ctr} are kept secret. □

Detection of data integrity violation. We want to make sure that any corruption to the outsourced data file $\widetilde{F}$ or the table BST_C on cloud servers can be detected. We prove this feature for $\widetilde{F}$ and the same ideas are applied to BST_C. The proof depends on the preimage and second-preimage resistance properties of the cryptographic hash function h.

Definitions.

1) **Preimage resistance**: given a hash value y, it is computationally infeasible to find any input x such that $h(x) = y$ [50]. Input x is called a preimage of y.
2) Second-preimage: given an input x, it is computationally infeasible to find a second input $x' \neq x$ such that $h(x) = h(x')$ [50].

Theorem 2. *Given a cryptographic hash function h with preimage and second-preimage resistance properties along with the non-collusion incentive of the TTP, any attempt to violate the integrity of outsourced data file on cloud servers will be detected.*

Proof. We prove the theorem by contradiction. The goal of a dishonest CSP is to send a corrupted or stale data file to authorized users without being detected. Let $\widetilde{D} = \{\tilde{d}_j\}_{1 \leq j \leq m}$ be the data file received by an authorized user from the CSP during the data access phase of the proposed scheme, where $\{\tilde{d}_j\}_{1 \leq j \leq m}$ denotes the file blocks. Let $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$ be the actual outsourced data file. The authorized user receives the authentic FH_{TTP} from the TTP, computes $FH_U = \oplus_{j=1}^{m} h(\tilde{b}_j)$, and checks $FH_U \stackrel{?}{=} FH_{TTP}$. If $FH_U \neq FH_{TTP}$, then $\widetilde{D} \neq \widetilde{F}$ (data has been corrupted on cloud servers).

For violating data integrity without being detected there are two possible scenarios. First, the CSP has to generate values $\{h_j^*\}_{1 \leq j \leq m}$ such that $FH_{TTP} = \oplus_{j=1}^{m} h_j^*$, and at least one $h_j^* \neq h(\tilde{b}_j)$. If the CSP could create $\widetilde{D} = \{\tilde{d}_j\}_{1 \leq j \leq m}$ such that $h_j^* = h(\tilde{d}_j)\ \forall j$, the cheating is possible. Due to the preimage-resistance property of h (one-way function), the CSP cannot generate such a data file $\widetilde{D}$, i.e., $\tilde{d}_j$ must be equal to $\tilde{b}_j\ \forall j$.

Second, the received data $\widetilde{D}$ has at least one block $\tilde{d}_j \neq \tilde{b}_j$, but $h(\tilde{d}_j) = h(\tilde{b}_j)\ \forall j$ to guarantee that $FH_U = FH_{TTP}$. Due to the second-preimage resistance property of h, there is no such a data file $\widetilde{D}$, i.e., $\tilde{d}_j$ must be equal to $\tilde{b}_j\ \forall j$. □

Remark 3. The encrypted block $\tilde{b}_j = E_{DEK}(\mathcal{BN}_j || b_j)$. Block number $\mathcal{BN}_j$ is embedded with the block b_j to be used along with BST_C to orderly reconstruct the plain file F if the blocks $\{b_j\}_{1 \leq j \leq m}$ are randomly received. Using a proof similar to that of Theorem 2, we can show that BST_C cannot be corrupted without being detected. Swapping the entries of BST_C without changing their contents can cause the file F to be reconstructed in an incorrect order. Although the CSP has no incentive and no financial benefit of doing such swapping, one can defend this weird behavior by storing the $\mathcal{BN}$ column of BST_O on the TTP side.

The authorized user can retrieve and use this column during the data access phase. This countermeasure adds little extra storage on the TTP ($4m$ bytes).

Assurance of newness property. Assurance of newness property is identical to detection of data integrity violation. With each dynamic operation (modification, insertion, deletion, append), the TTP update the values FH_{TTP} and TH_{TTP} to reflect the most recent state of the outsourced data. Thus, the CSP cannot respond to an access request by sending stale data without being detected.

Enforcement of access control. The proposed scheme combines the techniques of broadcast encryption key rotation, and lazy revocation to enforce access control of outsourced data.

Theorem 3. *Users can be granted or revoked access to outsourced data.*

Proof (Sketch). The owner creates $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$ and encrypts the outsourced data using $DEK = h(K_{ctr})$. Broadcast encryption bENC allows the data owner to encrypt the key K_{ctr} for an arbitrary subset of a group of users. According to the security strength of bENC [45], the users in the subset are only allowed to decrypt K_{ctr} and access the outsourced data.

It is acceptable for revoked users to access unmodified data blocks. However, updated/new blocks must be inaccessible by such revoked users. In case of data modification/insertion following a revocation, the data owner rolls K_{ctr} forward: $K_{ctr+1} = K_{ctr}^d \bmod N$, and then increments ctr by 1 (preparing for next rotation). Based on the hardness of the RSA problem [49], revoked users who possess K_l ($l < ctr$) are not able to generate K_{ctr}. Thus, combining broadcast encryption, key rotation, and lazy revocation achieves access control in the proposed scheme.

Detection of dishonest owner/user. The CSP signs both the file $\widetilde{F}$ and the table BST_C. The generated signatures σ_F and σ_T are sent along with $\widetilde{F}$ and BST_C to an authorized user during the data access phase. If the signature scheme is existentially unforgeable, the owner/user cannot falsely accuse the CSP regarding data integrity; the TTP performs signature verifications if there is a claim of data corruption. Recall that the signatures σ_F and σ_T are accepted by the authorized user as valid signatures in the beginning of the data access phase (step 3 in Fig. 6).

7 Performance Analysis

The data file F used in our performance analysis is of size 1GB with 4KB block size. Without loss of generality, we assume that the desired security level is 128-bit. Thus, we utilize a cryptographic hash h of size 256 bits (e.g., SHA-256), an elliptic curve defined over Galois field $GF(p)$ with $|p|$ = 256 bits (used for bENC), and BLS (Boneh-Lynn-Shacham) signature [51] of size 256 bits (used to compute σ_F and σ_T).

Here we evaluate the performance of the proposed scheme by analyzing the storage, communication, and computation overheads. We investigate overheads that the proposed scheme brings to a cloud storage system for *static* data with only *confidentiality* requirement. This investigation demonstrates whether the

features of our scheme come at a reasonable cost. The computation overhead is estimated in terms of the used cryptographic functions, which are notated in Table 2.

TABLE 2: Notation of cryptographic functions

Notation	Description
h	Cryptographic hashing
$\mathcal{FR}$	Forward key rotation
$\mathcal{BR}$	Backward key rotation
$\mathcal{S}_\sigma$	Signature genration
$\mathcal{V}_\sigma$	Signature verification
E_{DEK}	Symmetric encryption using the key DEK
bENC^{-1}	Decryption of bENC

Let m and n denote the number of file blocks and the total number of system users, respectively. Table 3 presents a theoretical analysis for the storage, communication, and computation overheads of the proposed scheme. Table 4 summarizes the storage and communication overheads for our data file F (1GB with 4KB block size) and 100,000 authorized users. The notation [*] indicates that the parameter * does not always exist in the overhead expression.

TABLE 3: Overhead analysis of the proposed scheme. The notation [*] indicates that the parameter * does not always exist in the overhead expression. † The $\oplus$ is not considered (negligible effect)

Overheads	Operations	Owner	User	CSP	TTP
Storage Overhead (in bytes)		$8m$	—	$8m$	68+32$\sqrt{n}$
Communication Overhead (in bytes)	Dynamic Operations	45 + [8 + 32$\sqrt{n}$]	—	—	—
	Data Access	—	—	64 + $8m$	68 + 32$\sqrt{n}$
Computation Overhead	Dynamic Operations	h + E_{DEK} + [$\mathcal{FR}$ + bENC]	—	—	2 h + [2 h] †
	Data Access	—	2 $\mathcal{V}_\sigma$ + 3m h + bENC^{-1} + [$\mathcal{BR}$] †	2 $\mathcal{S}_\sigma$	—
	Cheating Detection				2 $\mathcal{V}_\sigma$ + [2m h] †

TABLE 4: Storage and communication overheads for the data file F (1GB with 4KB block size) and 100,000 authorized users. ‡ Storage overhead is independent of F.

Overheads	Operations	Owner	User	CSP	TTP
Storage Overhead		2MB	—	2MB	≈ 10KB ‡
Communication Overhead	Dynamic Operations	45 bytes + [≈ 10KB]	—	—	—
	Data Access	—	—	≈ 2MB	≈ 10KB

7.1 Comments

Storage overhead is the additional storage space used to store necessary information other than the outsourced file $\widetilde{F}$. The overhead on the owner side is due to storing BST_O. An entry of BST_O is of size 8 bytes (two integers), and the total number of entries equals the number of file blocks m. During implementation $\mathcal{SN}$ is not needed to be stored in BST_O; $\mathcal{SN}$ is considered to be the entry/table index (BST_O is implemented as a linked list). The size of BST_O for the file F is only 2MB (0.2% of F). BST_O size can be further reduced if the file F is divided into larger blocks (e.g., 16KB). Like the owner, the storage overhead on the CSP side comes from the storage of BST_C. To resolve disputes that may arise regarding data integrity or newness property, the TTP stores FH_{TTP} and TH_{TTP}, each of size 256 bits. Besides, the TTP stores $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$ that enables the data owner to enforce access control for the outsourced data. ctr is 4 bytes, and bENC has storage complexity $O(\sqrt{n})$, which is practical for an organization (data owner) with n = 100,000 users. A point on the elliptic curve used to implement bENC can be represented by 257 bits (≈ 32 bytes) using compressed representation [52]. Therefore, the storage overhead on the TTP side is close to 10KB, which is independent of the outsourced file size. Overall, the storage overhead of the proposed scheme for the file F is less than 4.01MB (≈ 0.4% of F).

The communication overhead is the additional information sent along with the outsourced data blocks. During dynamic operations, the communication overhead on the owner side comes from the transmission of a block operation BlockOP (can be represented by 1 byte), a table entry $\text{TEntry}_{\mathsf{BlockOP}}$ (8 bytes), and a block index j (4 bytes). If a block is to be modified following a revocation process, $\mathcal{KV}_j$ (4 bytes) is sent to the TTP. Moreover, in case of a block modification/deletion, the owner sends a hash (32 bytes) of the block to be modified/deleted to the TTP for updating FH_{TTP}. Recall that the owner also sends Rot (4 + $32\sqrt{n}$ bytes) to the TTP if block modifications/insertions are to be performed following user revocations. Therefore, in the worst case scenario (i.e., block modifications following revocations), the owner's overhead is less than 10KB. The Rot represents the major factor in the communication overhead, and thus the overhead is only 45 bytes if block modification/deletion operations are to be preformed without revocations (only 13 bytes for insertion operations). In practical applications, the frequency of

dynamic requests to the outsourced data is higher than that of user revocations. Hence, the communication overhead due to dynamic changes on the data is about 1% of the block size (the block is 4KB in our analysis).

As a response to access the outsourced data, the CSP sends the file along with σ_F (32 bytes), σ_T (32 bytes), and BST_C ($8m$ bytes). Moreover, the TTP sends FH_{TTP} (32 bytes), TH_{TTP} (32 bytes), and Rot. Thus, the communication overhead due to data access is 64 + $8m$ bytes on the CSP side, and 68 + $32\sqrt{n}$ bytes on the TTP side. Overall, to access the file F, the proposed scheme has communication overhead close to 2.01MB ($\approx$ 0.2% of F).

A cloud storage system for static data with only confidentiality requirement has computation cost for encrypting the data before outsourcing and decrypting the data after being received from the cloud servers. For the proposed scheme, the computation overhead on the owner side due to dynamic operations (modification/insertion) comes from computing $DEK = h(K_{ctr})$ and encrypting the updated/inserted block, i.e., the overhead is one hash and one encryption operations. If a block modification/insertion operation is to be performed following a revocation of one or more users, the owner performs $\mathcal{FR}$ to roll K_{ctr} forward, and bENC to generate the Rot. Hence, the computation overhead on the owner side for the dynamic operations is h + E_{DEK} + $\mathcal{FR}$ + bEnc (worst case scenario). Updating BST_O and BST_C is done without usage of cryptographic operations (add, remove, or modify a table entry).

To reflect the most recent version of the outsourced data, the TTP updates the values FH_{TTP} and TH_{TTP}. If no revocation has been performed before sending a modify request, only FH_{TTP} is updated on the TTP side. Therefore, the maximum computation overhead on the TTP side for updating both FH_{TTP} and TH_{TTP} is $4\,h$.

Before accessing the data received from the CSP, the authorized user verifies two signatures (generated by the CSP), BST_C entries, and the data file. These verifications cost $2\,\mathcal{V}_\sigma + 2m\,h$. Moreover, the authorized user decrypts $\mathsf{bENC}(K_{ctr})$ part in the Rot to get K_{ctr}. For each received block, K_{ctr} is rotated backward to obtain the actual key that is used to decrypt the data block. The optimized way of key rotation (using the set Q) highly affects the performance of data access; many blocks need a few or no rotations. Moreover, one hash operation is performed per block to compute DEK. Overall, the computation overhead due to data access is $2\,\mathcal{V}_\sigma$ + $3m\,h$ + bENC^{-1} + [$\mathcal{BR}$] on the owner side, and $2\,\mathcal{S}_\sigma$ on the CSP side.

For determining a dishonest party, the TTP verifies σ_T and σ_F. In case of valid signatures, the TTP proceeds to compute TH_{temp} and FH_{temp}. The values TH_{temp} and FH_{temp} are compared with TH_{TTP} and FH_{TTP}, respectively. Hence, the *maximum* computation overhead on the TTP side due to cheating detection is $2\,\mathcal{V}_\sigma$ + $2m\,h$.

8 IMPLEMENTATION AND EXPERIMENTAL EVALUATION

8.1 Implementation

We implement the proposed scheme on top of Amazon Elastic Compute Cloud (Amazon EC2) [53] and Amazon Simple Storage Service (Amazon S3) [54] cloud platforms.

Amazon EC2 is a web service that enables customers to lunch and manage Linux/Unix and Windows server instances (virtual servers) in Amazon's data centers. Amazon EC2 instance can contain applications, libraries, data, and associated configuration settings. Amazon Web Services (AWS) Management Console is used to start, stop, reboot, scale, terminate, and monitor EC2 instances. Customers can automatically scale up and down the number of EC2 instances according to their demands. Moreover, customers can upgrade and downgrade a specific EC2 instance to fit current requirements.

Amazon S3 is storage for the Internet. It provides a simple web services interface that can be used to store and retrieve almost unlimited amount of data. Amazon S3 stores data as objects within buckets. A bucket can contain unlimited number of objects, where an object is compromised of a data file (any kind: a text file, a photo, a video) and optionally any metadata that describes that file. The object can contain from 1 byte to 5 terabytes of data. Customers are allowed to choose the geographic locations where Amazon S3 will store the bucket and its contents. Using AWS Management Console, we can create /delete buckets and upload/delete data files. Moreover, folders can be created inside a bucket to group objects.

The proposed scheme consists of four modules: `OModule` (owner module), `CModule` (CSP module), `UModule` (user module), and `TModule` (TTP module). `OModule`, which runs on the owner side, is a library to be used by the owner to perform the owner role in the setup and file preparation phase. Moreover, this library is used by the owner during the dynamic operations on the outsourced data. `CModule` is a library that runs on Amazon EC2 and is used by the CSP to store, update, and retrieve data from Amazon S3. `UModule` is a library to be run at the authorized users' side, and include functionalities that allow users to interact with the TTP and the CSP to retrieve and access the outsourced data. `TModule` is a library used by the TTP to perform the TTP role in the setup and file preparation phase. Moreover, the TTP uses this library during the dynamic operations and to determine the cheating party in the system.

Implementation settings. In our implementation we use a "large" Amazon EC2 instance to run `CModule`. This instance type provides total memory of size 7.5 GB 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 - 1.2 GHz 2007 Opteron or 2007 Xeon processor [55]. The other three modules are executed on a desktop computer with Intel(R) Xeon (R) 2 GHz processor and 3GB RAM running Windows XP. We outsource a data file of size 1GB to Amazon S3. Algorithms (hashing, broadcast encryption, digital signatures, etc.) are implemented using MIRACL library version 5.5.4. For a 128-bit security level, bENC uses an elliptic curve with a 256-bit group order. In the experiments, we utilize SHA-256, 256-bit BLS signature,

and Barreto-Naehrig (BN) [56] curve defined over prime field $GF(p)$ with $|p|$ = 256 bits and embedding degree = 12 (the BN curve with these parameters is provided by the MIRACL library).

8.2 Experimental Evaluation

In this section we experimentally evaluate the computation overhead the proposed scheme brings to a cloud storage system that has been dealing with static data with only confidentiality requirement.

To experimentally evaluate the computation overhead on the owner side due to the dynamic operations, we perform 100 different block operations with number of authorized users ranging from 20,000 to 100,000. We run our experiment three times, each time with a different revocation percentage. In the first time, 5% of 100 dynamic operations are executed following revocations. We increase the revocation percentage to 10% for the second time and 20% for the third time. Fig. 8 shows the owner's average computation overhead per operation. For a large organization (data owner) with 100,000 users, performing dynamic operations and enforcing access control with 5% revocations add about 63 milliseconds of overhead. With 10% and 20% revocation percentages, which are high percentages than an average value in practical applications, the owner overhead is 0.12 and 0.25 seconds, respectively.

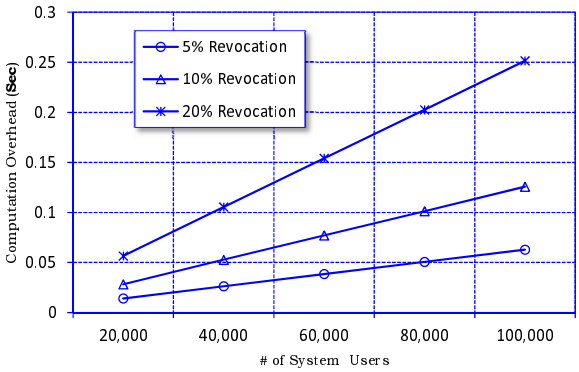

Fig. 8: Owner's average computation overhead due to dynamic operations.

Scalability (i.e., how the system performs when more users are added) is an important feature of cloud storage systems. The access control of the proposed scheme depends on the square root of the total number of system users. Fig. 8 shows that for a large organization with 100,000 users, performing dynamic operations and enforcing access control for outsourced data remains practical.

Table 5 shows the computation overheads of the proposed scheme on the TTP, the CSP, and the authorized users sides.

In the worst case, the TTP executes only 4 hashes per dynamic request to reflect the change on the outsourced data. Thus, the maximum computation overhead on the TTP side is about 0.08 milliseconds, i.e., the proposed scheme brings light overhead on the TTP during the normal system operations.

TABLE 5: Experimental results of the computation overheads

Component	TTP	Authorized Users	CSP
Computation Overhead	0.08 ms / 6.46 s	0.6 s	6.04 s

To identify the dishonest party in the system in case of disputes, the TTP verifies two signatures (σ_F and σ_T), computes combined hashes for the data (file and table), and compare the computes hashes with the authentic values (TH_{TTP} and FH_{TTP}). Thus, the computation overhead on the TTP side is about 6.46 seconds. Through our experiments, we use only one desktop computer to simulate the TTP and accomplish its work. The TTP may choose to split the work among a few devices or use a single device with a multi-core processor which is becoming prevalent these days, and thus the computation time on the TTP side is significantly reduced in many applications.

The computation overhead on the user side due to data access comes from five aspects divided into two groups. The first group involves signatures verification and hash operations to verify the received data (file and table). The second group involves broadcast decryption, backward key rotations, and hash operations to compute the DEK. The first group costs about 6.46 seconds, which can be easily hidden in the receiving time of the data (1GB file and 2MB table).

To investigate the computation time of the second group, we access the file after running 100 different block operations (with 5% and 10% revocation percentages). Moreover, we implement the backward key rotations in the optimized way. The second group costs about 0.6 seconds, which can be considered as the user's computation overhead due to data access.

As a response to the data access request, the CSP computes two signatures: σ_F and σ_T. Thus, the computation overhead on the CSP side due to data access is about 6.45 seconds and can be easily hidden in the transmission time of the data (1GB file and 2MB table).

9 Conclusions

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 different aspects of outsourcing data storage: block-level dynamic operations, newness, mutual trust, and access control.

We have proposed a cloud-based storage scheme which supports outsourcing of dynamic data, where the owner is capable of not only archiving and accessing the data stored by the CSP, but also updating and scaling this data on the remote servers. The proposed scheme enables the authorized users to ensure that they are receiving the most recent version of the outsourced data. Moreover, in case of dispute regarding data integrity/newness, a TTP is able to determine the dishonest party. The data owner enforces access control for the outsourced data by combining three cryptographic techniques: broadcast encryption, lazy revocation, and key rotation. We have studied the security features of the proposed scheme.

In this paper, we have investigated the overheads added by the proposed scheme when incorporated into a cloud storage model for *static* data with only *confidentiality* requirement. The storage overhead is $\approx$ 0.4% of the outsourced data size, the communication overhead due to block-level dynamic changes on the data is $\approx$ 1% of the block size, and the communication overhead due to retrieving the data is $\approx$ 0.2% of the outsourced data size. For a large organization (data owner) with 100,000 users, performing dynamic operations and enforcing access control add about 63 milliseconds of overhead. Therefore, important features of outsourcing data storage can be supported without excessive overheads in storage, communication, and computation.

ACKNOWLEDGMENTS

This work was supported in part through an NSERC grant awarded to Prof. Hasan.

REFERENCES

[1] A. Singh and L. Liu, "Sharoes: A data sharing platform for outsourced enterprise storage environments," in *Proceedings of the 24th International Conference on Data Engineering, ICDE*. IEEE, 2008, pp. 993–1002.

[2] 104th United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Online at http://aspe.hhs.gov/admnsimp/pl104191.htm, 1996.

[3] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2007, pp. 598–609.

[4] Y. Deswarte, J.-J. Quisquater, and A. Saïdane, "Remote integrity checking," in *6th Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, S. J. L. Strous, Ed., 2003, pp. 1–11.

[5] D. L. G. Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer," Cryptology ePrint Archive, Report 2006/150, 2006.

[6] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *Trans. Storage*, vol. 2, no. 2, 2006.

[7] 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. on Knowl. and Data Eng.*, vol. 20, no. 8, 2008.

[8] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, Berkeley, CA, USA, 2007, pp. 1–6.

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

[10] K. Zeng, "Publicly verifiable remote data integrity," in *Proceedings of the 10th International Conference on Information and Communications Security*, ser. ICICS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 419–434.

[11] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *SecureComm '08: Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, New York, NY, USA, 2008, pp. 1–10.

[12] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, New York, NY, USA, 2009, pp. 213–222.

[13] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Transactions on Knowledge and Data Engineering*, vol. 99, no. PrePrints, 2011.

[14] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," Cryptology ePrint Archive, Report 2009/081, 2009, http://eprint.iacr.org/.

[15] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *ESORICS'09: Proceedings of the 14th European Conference on Research in Computer Security*, Berlin, Heidelberg, 2009, pp. 355–370.

[16] A. F. Barsoum and M. A. Hasan, "Provable possession and replication of data over cloud servers," Centre For Applied Cryptographic Research (CACR), University of Waterloo, Report 2010/32, 2010, http://www.cacr.math.uwaterloo.ca/techreports/2010/cacr2010-32.pdf.

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

[18] Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in *Second International Symposium on Data, Privacy, and E-Commerce*, 2010.

[19] A. F. Barsoum and M. A. Hasan, "On verifying dynamic multiple data copies over cloud servers," Cryptology ePrint Archive, Report 2011/447, 2011, 2011, http://eprint.iacr.org/.

[20] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: a high-availability and integrity layer for cloud storage," in *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2009, pp. 187–198.

[21] ——, "Proofs of retrievability: theory and implementation," in *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*. New York, NY, USA: ACM, 2009, pp. 43–54.

[22] R. Curtmola, O. Khan, and R. Burns, "Robust remote data checking," in *StorageSS '08: Proceedings of the 4th ACM international workshop on Storage security and survivability*. New York, NY, USA: ACM, 2008, pp. 63–68.

[23] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *TCC '09: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 109–127.

[24] A. Juels and B. S. Kaliski, "PORs: Proofs of Retrievability for large files," in *CCS'07: Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 584–597.

[25] H. Shacham and B. Waters, "Compact proofs of retrievability," Cryptology ePrint Archive, Report 2008/073, 2008, http://eprint.iacr.org/.

[26] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proceedings of the FAST 03 Conference on File and Storage Technologies*. USENIX, 2003.

[27] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *Proceedings of the Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2003.

[28] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *Proceedings of the Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2005.

[29] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: Management of access control evolution on outsourced data," in *Proceedings of the 33rd International Conference on Very Large Data Bases*. ACM, 2007, pp. 123–134.

[30] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS '06. ACM, 2006, pp. 89–98.

[31] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, ser. SP '07. IEEE Computer Society, 2007, pp. 321–334.

[32] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. IEEE Press, 2010, pp. 534–542.

[33] ——, "Attribute based data sharing with attribute revocation," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10. ACM, 2010, pp. 261–270.

[34] S. Narayan, M. Gagné, and R. Safavi-Naini, "Privacy preserving EHR system using attribute-based infrastructure," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, ser. CCSW '10. ACM, 2010, pp. 47–52.

[35] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, "Enabling security in cloud storage SLAs with cloudproof," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, ser. USENIXATC'11. USENIX Association, 2011.

[36] D. Naor, M. Naor, and J. B. Lotspiech, "Revocation and tracing schemes for stateless receivers," in *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '01. Springer-Verlag, 2001, pp. 41–62.

[37] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *EUROCRYPT*, 1998, pp. 127–144.

[38] M. J. Atallah, K. B. Frikken, and M. Blanton, "Dynamic and efficient key management for access hierarchies," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, ser. CCS '05. ACM, 2005, pp. 190–202.

[39] W. Wang, Z. Li, R. Owens, and B. Bhargava, "Secure and efficient access to outsourced data," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, ser. CCSW '09. ACM, 2009, pp. 55–66.

[40] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *INFOCOM*, 2010, pp. 525–533.

[41] K. E. Fu, "Group sharing and random access in cryptographic storage file systems," Master's thesis, MIT, Tech. Rep., 1999.

[42] M. Backes, C. Cachin, and A. Oprea, "Secure key-updating for lazy revocation," in *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science. Springer, 2006, pp. 327–346.

[43] E. Riedel, M. Kallahalla, and R. Swaminathan, "A framework for evaluating storage system security," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. USENIX Association, 2002.

[44] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*. CRC Press, Inc., 1997.

[45] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Advances in Cryptology - CRYPTO*, 2005, pp. 258–275.

[46] A. Fiat and M. Naor, "Broadcast encryption," in *Proceedings of the 13th annual international cryptology conference on Advances in cryptology*. Springer-Verlag New York, Inc., 1994, pp. 480–491.

[47] A. Menezes, "An introduction to pairing-based cryptography," Lecture Notes 2005, Online at http://www.math.uwaterloo.ca/~ajmeneze/publications/pairings.pdf.

[48] "FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November 2001." [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[49] D. Boneh and R. Venkatesan, "Breaking RSA may not be equivalent to factoring," in *EUROCRYPT*, 1998, pp. 59–71.

[50] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *FSE: Fast Software Encryption*, 2004, pp. 371–388.

[51] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, London, UK, 2001, pp. 514–532.

[52] P. S. L. M. Barreto and M. Naehrig, "IEEE P1363.3 submission: Pairing-friendly elliptic curves of prime order with embedding degree 12," New Jersey: IEEE Standards Association, 2006.

[53] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/.

[54] Amazon Simple Storage Service (Amazon S3), http://aws.amazon.com/s3/.

[55] Amazon EC2 Instance Types, http://aws.amazon.com/ec2/.

[56] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proceedings of SAC 2005, volume 3897 of LNCS*. Springer-Verlag, 2005, pp. 319–331.