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

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

Abstract—Storage-as-a-Service offered by cloud service providers (CSPs) is a *paid* facility that enables organizations to outsource their *sensitive* data to be stored on remote servers. 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

In the current era of digital world, various organizations produce a large amount of sensitive data including personal information, electronic health records, and financial data. The local management of such huge amount of data is problematic and costly due to the requirements of high storage capacity and qualified personnel. Therefore, Storage-as-a-Service offered by cloud service providers (CSPs) emerged as a solution to mitigate the burden of large local data storage and reduce the maintenance cost by means of outsourcing data storage.

Since the data owner physically releases sensitive data to a remote CSP, there are some concerns regarding *confidentiality*, *integrity*, and *access control* of the data. 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 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 data, e.g., [1]–[8]. Proof of retrievability [9]–[12] 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 [13]–[16], which aim at providing data storage security on untrusted remote servers. Another class of solutions utilizes attribute-based encryption to achieve fine-grained access control [17], [18].

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 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 [19].

In this work, we propose a scheme that addresses important issues related to outsourcing the storage of data, namely *dynamic data*, *newness*, *mutual trust*, and *access control*. The remotely stored data can be not only accessed by authorized users, but also updated and scaled by the owner. After updating, 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. 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 and the responsible party is identified. Last but not least, the access control is considered, which allows the owner to grant or revoke access rights to the outsourced data. Appendix A discusses existing research related to our work.

Main contributions:

- The design and implementation of a cloud-based storage scheme that has the following features: (i) it allows a data owner to outsource the data to a CSP, and perform full dynamic operations at the block-level, i.e., it supports operations such as block modification, insertion, deletion, and append; (ii) it ensures the newness property, i.e., the authorized users receive the most recent version of the outsourced data; (iii) it establishes *indirect* mutual trust between the data owner and the CSP since each party resides in a different trust domain; and (iv) it enforces the access control for the outsourced data.
- 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.

2 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 capabilities to detect/specify dishonest parties.

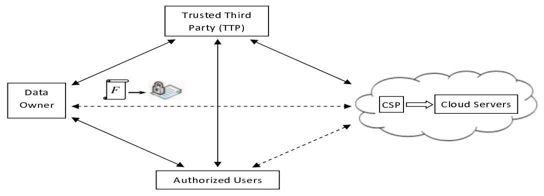


Fig. 1: Cloud computing data storage system model.

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 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.

Remark 1. In this work, the auditing process of the data received from the CSP is done by 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. 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. In addition, the owner enforces access control by granting or revoking access rights to the outsourced data. To access the data, the 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).

The TTP is an independent entity, and thus has no incentive to collude with any party. 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.

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, 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, 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.

3 System Preliminaries

3.1 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 *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 [20]. 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 [19], [21], [22].

3.2 Key Rotation

Key rotation [13] 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 proposed scheme in this work utilizes the key rotation technique [13]. 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 $ed \equiv 1 \text{ mod } (p-1)(q-1)$.

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 as $K_{ctr+1} = K_{ctr}^d$ mod N. Authorized users can recursively generate older versions of the current key as $K_{ctr-1} = K_{ctr}^e$ mod N (*rotating backward*).

3.3 Broadcast Encryption

Broadcast encryption (bENC) 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. The proposed scheme uses bENC [23] to enforce access control in outsourced data. The bENC [23] 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 $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. 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 $\{d_i\}_{1 \le i \le n}$, 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 [23], 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 [23] 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 $\mathbb{G}$.

4 Proposed Cloud-Based Storage Scheme

4.1 Warmup Discussion

Before presenting our main scheme, we discuss a straightforward solution using authentication tags (digital signatures) to detect cheating from any side (data owner or CSP). For a file $F = \{b_j\}_{1 \le j \le m}$, the owner attaches a tag $\texttt{OWN}\sigma_j$ with each block before outsourcing. The owner sends $\{b_j, \texttt{OWN}\sigma_j\}_{1 \le j \le m}$ to the CSP, where the tags $\{\texttt{OWN}\sigma_j\}_{1 \le j \le m}$ 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 $\{\texttt{OWN}\sigma_j\}_{1 \le j \le m}$ 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 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 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 on the cloud servers. Moreover, there is an increased computation overhead on different system components; for a file F containing m blocks, it requires $2m$ signature generations and $3m$ signature verifications, which may be computationally a challenging task for large data files. A file is of size 1GB with 4KB block size requires 2^{19} signature generations and 3×2^{18} signature verifications.

If the CSP receives the data blocks from a trusted entity, 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.

4.2 Overview and Rationale

Validating the outsourced 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.

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.

4.3 Notations

- $F = \{b_1, b_2, \ldots, b_m\}$ is a data file
- 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 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
- $Rot = \langle ctr, \text{bENC}(K_{ctr})\rangle$ is a rotator, where $\text{bENC}(K_{ctr})$ is a broadcast encryption of K_{ctr}
- $\oplus$ is an XOR operator

4.4 Block Status Table

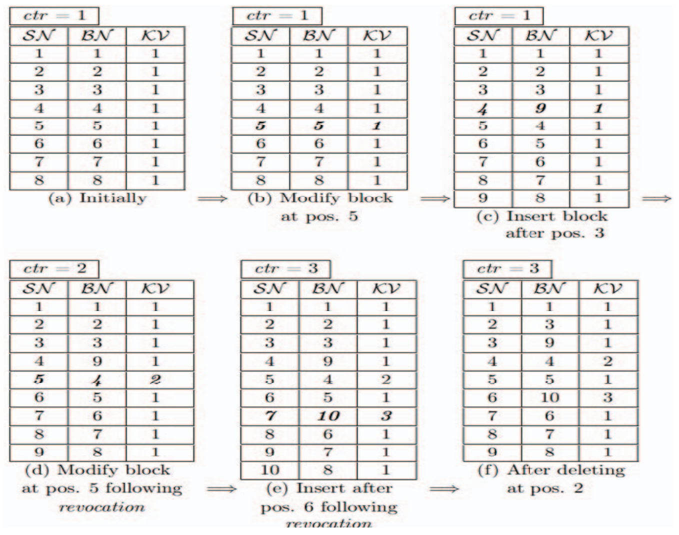
The block status table (BST) is a small *dynamic* data structure used to reconstruct and access file blocks outsourced to the CSP. The BST consists of three columns: serial number ($\mathcal{SN}$), block number ($\mathcal{BN}$), and key version ($\mathcal{KV}$). $\mathcal{SN}$ is an indexing to the file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



$ctr = 1$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1

(a) Initially $\Longrightarrow$

$ctr = 1$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	4	1
5	***5***	***1***
6	6	1
7	7	1
8	8	1

(b) Modify block at pos. 5 $\Longrightarrow$

$ctr = 1$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	***9***	***1***
5	4	1
6	5	1
7	6	1
8	7	1
9	8	1

(c) Insert block after pos. 3 $\Longrightarrow$

$ctr = 2$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	9	1
5	***4***	***2***
6	5	1
7	6	1
8	7	1
9	8	1

(d) Modify block at pos. 5 following *revocation* $\Longrightarrow$

$ctr = 3$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	9	1
5	4	2
6	5	1
7	***10***	***3***
8	6	1
9	7	1
10	8	1

(e) Insert after pos. 6 following *revocation* $\Longrightarrow$

$ctr = 3$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	3	1
3	9	1
4	4	2
5	5	1
6	10	3
7	6	1
8	7	1
9	8	1

(f) After deleting at pos. 2

Fig. 2: Changes in the BST due to different dynamic operations on a file $F = \{b_j\}_{1 \leq j \leq 8}$.

4.5 Procedural Steps of the Proposed Scheme

4.5.1 Setup and File Preparation

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 $F = \{b_j\}_{1 \leq j \leq m}$, 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 $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$, 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 supports in reconstructing the file blocks in the correct order (more details will be explained later).

TTP Role. To resolve disputes that may arise regarding data integrity/newness, the TTP computes combined hash values for the encrypted file $\widetilde{F}$ and the BST. It 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 authorized users to reconstruct and access the outsourced data file. The proposed scheme assumes that the owner is *intermittently* online and the users are enabled to access the data even when the owner is offline. To this end, the CSP stores a copy of the BST along with the outsourced data file. When a 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. 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 system.

Owner	TTP	CSP
ctr, K_{ctr}, BST$_O$	Rot, FH_{TTP}, TH_{TTP}	$\widetilde{F}$, BST$_C$

4.5.2 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 $\langle \mathsf{BlockOp}, \text{TEntry}_{\mathsf{BlockOp}}, j, \mathcal{KV}_j, h(\tilde{b}_j), \texttt{RevFlag}, b^* \rangle$, where BlockOp corresponds to block modification (denoted by BM), block insertion (denoted by BI), or block deletion (denoted by BD). 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. 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-

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

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 value TH_{TTP}.

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 $b_{j+1} = \bar{b}$. Fig. 4 describes the steps performed by each system component (owner, CSP, and TTP) during block insertion.

Append. It means adding a new block at the end of the outsourced data. It can simply be implemented via insert operation after the last block of the data file.

Deletion. When one block is deleted all subsequent blocks are moved one step forward. 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 $\tilde{b}_j$ from the combined hash FH_{TTP} (properties of $\oplus$ operator). The same idea is used with the TH_{TTP} value.

4.5.3 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$_C$, respectively. The 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 $TH_U = \oplus_{j=1}^{m} h(\mathcal{BN}_j || \mathcal{KV}_j)$, and comparing it with the authentic value TH_{TTP} received from the TTP. If the 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 user continues to verify the contents of the file $\widetilde{F}$ by computing $FH_U = \oplus_{j=1}^{m} h(\tilde{b}_j)$ and comparing 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 $\tilde{b}_j$. The component bENC(K_{ctr}) of Rot is decrypted to get the most recent key K_{ctr}. Using the key rotation technique, the 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 $\mathcal{KV}_j$ can determine how many rotation steps for K_{ctr} with each block $\tilde{b}_j$. Decrypting the block $\tilde{b}_j$ returns $(\mathcal{BN}_j || b_j)$. Both $\mathcal{BN}_j$ 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.

```
/* 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 $Rot = \langle ctr, \text{bENC}(K_{ctr}) \rangle$
       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 entry TEntry$_{\text{BM}} = \{\mathcal{BN}_j, \mathcal{KV}_j\}$
  4) Sends a modify request $\langle$BM, TEntry$_{\text{BM}}$, $j$, $\overline{\mathcal{KV}}_j$, $h(\tilde{b}_j)$,
     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 $\tilde{b}_j$ 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 BST$_C$ entry at index $j$ using TEntry$_{\text{BM}}$

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.

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 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 $i - \mathcal{KV}_j$. Then, K_i is 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 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

```
/* 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 $\text{TEntry}_{\mathsf{BI}} = \{\mathcal{BN}_{j+1}, \mathcal{KV}_{j+1}\} = \{1 + Max\{\mathcal{BN}_j\}_{1\le j\le m},\ ctr\}$, and
     inserts this entry in $\text{BST}_O$ 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 a request $\langle$BI, $\text{TEntry}_{\mathsf{BI}}$, $j$, $null$, $null$, 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 $\text{BST}_C$

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 $Rot$ with the newly received value
```

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

```
/* Deletion of a block $b_j$ from the outsourced file */
Data Owner
  1) Copies the entry at index $j$ from $\text{BST}_O$ to a block-delete table entry $\text{TEntry}_{\mathsf{BD}} = \{\mathcal{BN}_j, \mathcal{KV}_j\}$
  2) Deletes the entry at index $j$ from $\text{BST}_O$
  3) Sends a request $\langle$BD, $\text{TEntry}_{\mathsf{BD}}$, $j$, $null$, $h(\tilde{b}_j)$, false, $null\rangle$ to both the CSP and the TTP (OSMR), 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 $\text{BST}_C$ 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 $\text{BST}_C$

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.

```
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 $\widetilde{F} = \{\tilde{b}_j\}_{1\le j\le m}$ associated with a signature $\sigma_F$ (CSP's signature on the entire file),
   and sending $\text{BST}_C$ associated with a signature $\sigma_T$ (CSP's signature on the entire table) to the authorized user
3) The authorized user verifies $\sigma_F$ and $\sigma_T$, and accepts the data only if $\sigma_F$ and $\sigma_T$ 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 $\text{BST}_C$ entries
     a) The user computes $TH_U = \oplus_{j=1}^{m} h(\mathcal{BN}_j||\mathcal{KV}_j)$
     b) If the user claims that $TH_U \ne TH_{TTP}$ 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 user claims that $FH_U \ne FH_{TTP}$ 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*/
            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 $\text{BST}_C$
          – 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.

```
Cheating Detection Procedure
The TTP is invoked to determine the dishonest party:
  1) The TTP verifies $\sigma_T$ and $\sigma_F$
  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 $TH_{temp} \ne TH_{TTP}$ or $FH_{temp} \ne FH_{TTP}$ 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.

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 $TH_{temp} \ne TH_{TTP}$ or $FH_{temp} \ne FH_{TTP}$, this indicates that the CSP is dishonest for sending corrupted data, otherwise the owner/user is dishonest for falsely claiming integrity violation of received data. The security analysis of the proposed scheme is given in Appendix B.

5 Performance Analysis

5.1 Settings and Overheads

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 [24] 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.

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.

TABLE 2: Notation of cryptographic functions.

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

5.2 Comments

Storage overhead. It 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. The 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 ($\approx$ 32 bytes) using compressed representation [25]. Therefore, the storage overhead on the TTP side is close to 10KB, which is independent of the outsourced file size. Overall, the storage overhead for the file F is less than 4.01MB ($\approx$ 0.4% of F).

Communication overhead. It 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 $\text{TEntry}_{\text{BlockOP}}$ (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).

Computation overhead. 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

TABLE 3: Overhead analysis of the proposed scheme. The overheads shown in square brackets are not always present and are incurred when revocation(s) actually occur.

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

‡ The cost of $\oplus$ is usually negligible and is omitted in the overhead expressions.

TABLE 4: Storage and communication overheads for the data file F (1GB with 4KB block size) and 100,000 authorized users. The values shown in square brackets are not always present and are incurred when revocation(s) actually occur.

Overheads	Operations	Owner	User	CSP	TTP
Storage		2MB	—	2MB	$\approx$ 10KB †
Communication	Dynamic Operations	45 bytes + [$\approx$ 10KB]	—	—	—
	Data Access	—	—	$\approx$ 2MB	$\approx$ 10KB

† Storage overhead is independent of F.

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 $2\,\mathcal{V}_\sigma + 3m\,h + \mathsf{bENC}^{-1} + [\mathcal{BR}]$ 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 $2\,\mathcal{V}_\sigma + 2m\,h$.

6 Implementation and Experimental Evaluation

6.1 Implementation

We have implemented the proposed scheme on top of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3) [26] cloud platforms. Our implementation of 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 7.5GB and 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each). One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0 - 1.2GHz 2007 Opteron or 2007 Xeon processor. A separate server in the lab is used to run `TModule`. This server has Intel(R) Xeon(TM) 3.6GHz processor, 2.75GB RAM, and Windows XP operating system. The `OModule` is executed on a desktop computer with Intel(R) Xeon(R) 2GHz processor and 3GB RAM running Windows XP. A laptop with Intel(R) Core(TM) 2.2GHz processor and 4GB RAM running Windows 7 is used to execute the `UModule`. 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) [27] curve defined over prime field $GF(p)$ with $|p|$ = 256 bits and embedding degree = 12.

6.2 Experimental Evaluation

Here we describe the experimental evaluation of the computation overhead the proposed scheme brings to a cloud storage system that has been dealing with static data with only confidentiality requirement.

Owner computation overhead. To experimentally evaluate the computation overhead on the owner side due to the dynamic operations, we have performed 100 different block operations (modify, insert, append, and delete) with number of authorized users ranging from 20,000 to 100,000. We have 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 increased 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.

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 10^5 users, performing dynamic operations and enforcing access control for outsourced data remains practical.

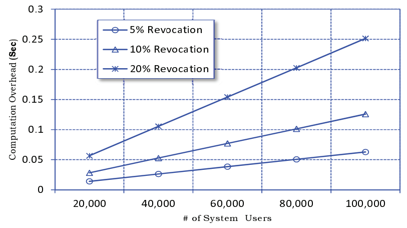


Fig. 8: Owner's average computation overhead due to dynamic operations.

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

TABLE 5: Experimental results of the computation overheads

Component	TTP	Users	CSP
Computation Overhead	0.04 ms / 3.59 s	0.55 s	6.04 s

TTP computation overhead. 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 0.04 milliseconds, i.e., the proposed scheme brings light overhead on the TTP during the normal system operations.

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 3.59 seconds. Through our experiments, we use only one server 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.

User computation overhead. 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 5.87 seconds, which can be easily hidden in the receiving time of the data (1GB file and 2MB table).

To investigate the 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 0.55 seconds, which can be considered as the user's computation overhead due to data access.

CSP computation overhead. 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 6.04 seconds and can be easily hidden in the transmission time of the data (1GB file and 2MB table).

7 Conclusions

In this paper, 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.

We have investigated the overheads added by our 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 with 10^5 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.

REFERENCES

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

[2] 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.

[3] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, 2008, pp. 1–10.

[4] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009, pp. 213–222.

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

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

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

[8] 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/.

[9] 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.

[10] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, 2009.

[11] 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.

[12] H. Shacham and B. Waters, "Compact proofs of retrievability," in *ASIACRYPT '08*, 2008, pp. 90–107.

[13] 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: File and Storage Technologies*, 2003.

[14] 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*, 2003.

[15] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *NDSS*, 2005.

[16] 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.

[17] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *CCS '06*, 2006, pp. 89–98.

[18] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *INFOCOM'10*, 2010, pp. 534–542.

[19] 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*, 2011.

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

[21] 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*, 2009, pp. 55–66.

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

[23] 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.

[24] 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.

[25] 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.

[26] Amazon Web Service, http://aws.amazon.com/.

[27] 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.

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

[29] 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.

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

[31] 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.

[32] J. Feng, Y. Chen, W.-S. Ku, and P. Liu, "Analysis of integrity vulnerabilities and a non-repudiation protocol for cloud data storage platforms," in *Proceedings of the 2010 39th International Conference on Parallel Processing*, 2010, pp. 251–258.

[33] J. Feng, Y. Chen, and D. H. Summerville, "A fair multi-party non-repudiation scheme for storage clouds," in *2011 International Conference on Collaboration Technologies and Systems*, 2011, pp. 457–465.

Ayad Barsoum is a Ph.D. Candidate in the Department of Electrical and Computer Engineering at University of Waterloo since 2009. He received his B.Sc. and M.Sc. degrees in computer science from Ain Shams University, Egypt, in 2000 and 2004, respectively. He is a member of the Centre for Applied Cryptographic Research at the University of Waterloo.

Barsoum was awarded the Ain Shams University Scholarship of Excellence four times. At the University of Waterloo, Barsoum has received the Graduate Research Studentship, the International Doctoral Award, and the University of Waterloo Graduate Scholarship.

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

Dr. Hasan joined the Department of Electrical and Computer Engineering, University of Waterloo (UW), Ontario, Canada in 1993 and has been a full professor since 2002. At UW, he is also a member of the Centre for Applied Cryptographic Research, the Center for Wireless Communications, and the VLSI Research group.

Dr. Hasan is a recipient of the Raihan Memorial Gold Medal. At the University of Victoria, he was awarded the President's Research Scholarship four times. At UW, he received a Faculty of Engineering Distinguished Performance Award in 2000, and Outstanding Performance Awards in 2004 and 2010. He served on the program and executive committees of several conferences. From 2000 to 2004, he was an associate editor of the IEEE Transactions of Computers. He is a licensed professional engineer of Ontario.