Enabling Indirect Mutual Trust And Dynamic Data In Cloud Computing

Chaitali Kharbade[1], Vishwajit Bajpai[2], Shyam P Dubey[3]

[1] *Nuva College of Engineering & Technology,Nagpur,* [2] *MIET,Gondia,* [3] *Nuva College of Engineering & Technology,Nagpur*

[1] chaitali2kharbade@gmail.com, [2] bajpayee07@gmail.com, [3] shyam.nuva@rediffmail.com

***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.**

Keywords **-Outsourcing data storage, dynamic environment, mutual trust, access control**

I. INTRODUCTION

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.

II. RELATED WORK

Many researchers have proposed storing encrypted data in the cloud to defend against the CSP [1], [2]. Under this approach, users are revoked by having a third party to reencrypt data such that previous keys can no longer decrypt any data [14]–[16]. The solution by [15] for instance, lets the data owner issue a re-encryption key to an untrusted server to reencrypt the data. Their solution utilizes PRE [6], which allows the server to re-encrypt the stored ciphertext to a different cipertext that can only be decrypted using a different key. During the process, the server does not learn the contents of the cipertext or the decryption keys. ABE is a new cryptographic technique that efficiently supports fine grained access control. The combination of PRE and ABE was first introduced by [9], and extended by [8], [17]. In [8], a hierarchical attribute-based encryption (HABE) scheme is proposed to achieve high performance and full delegation. The main difference between prior work and ours is that we do not require the underlying cloud infrastructure to be reliable in order to ensure correctness. Our scheme relies on time to re-encrypt data. However, in a cloud, the internal clock of each cloud server may differ. There have been several solutions to this problem. For instance, [10] proposed a probabilistic synchronization scheme, which exchanges messages to get remote servers' accurate clocks with high probability. Work by [11] used message delay to estimate the maximal difference between two communicating nodes to synchronize the clocks. Work by [13] proposed a clock synchronization scheme for cloud environments, which uses an authoritative time source shared by all participants in a transaction to achieve clock synchronization between virtual cloud policy enforcement points. By applying these techniques to achieve loose synchronization in the cloud, and to determine the maximal time difference between the data owner and each cloud server, our R3 scheme can always achieve correct access.

A. *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.

III. PRILIMINARIES

We consider a cloud computing environment consisting of a data owner, a cloud service provider (CSP) and multiple data users.

A. Problem Formulation

The data owner outsources his data in the form of a set of files *F*1*; • • • ; Fn* to the CSP. Each file is encrypted by the data owner before uploading to the CSP. Data users that want to access a particular file must first obtain the necessary keys from the data owner in order to decrypt the file. The data owner can also update the contents of a file after uploading it to the CSP. This is termed a *write* command. Each file, *F*, is encrypted with two parameters, *time slice* and *attributes*. We divide time into time slices, and every time slice is of equal length. We denote a particular time slice, *TS*, with a subscript, where *TSi* = [*ti; ti*+1).illustrates this concept. Attributes are organized into an *access structure*, A, which regulates access to a file. For example, a file with attributes _1*;* _2*;* _3 and A = *{*(_1∧_2)∨_3*}*, requires either both attributes _1 and _2, or just _3, to satisfy the access structure. A file *F* can only be decrypted with keys that satisfy *both* the access structure and time slice. A data user, after being authenticated by the data owner, is granted a set of keys, each of which is associated with an *attribute* and an *effective time* that denotes the length of time the user is authorized to possess the attributes. For example, if Alice is authorized to possess attributes *a*1*; : : : ; am* from *TS*1 to *TSn*, she will be issued keys as is shown in Table I. The security requirements of the R3 scheme are as follows:

1) Access control correctness. This requires that a data user with invalid keys cannot decrypt the file.
2) Data consistency. This requires that all data users who request file *F*, should obtain the same content in the same time slice.
3) Data confidentiality. The file content can only be known to data users with valid keys. The CSP is not considered a valid data user.
4) Efficiency. The cloud servers should not re-encrypt any file unnecessarily. This means that a file that has not been requested by any data user should not be re-encrypted.

TABLE I
ALICE'S KEYS

Key	Description
$SK^1_{a_1}$	Keys for attributes a_1 for TS_1
...	...
$SK^1_{a_m}$	Keys for attributes a_m for TS_1
...	...
$SK^n_{a_1}$	Keys for attributes a_1 for TS_n
...	...
$SK^n_{a_m}$	Keys for attributes a_m for TS_n

B. *Adversary Model*

Our system considers two types of adversaries. The first type of adversary is the CSP. The CSP adversary is considered honest-but-curious.

TABLE II
SUMMARY OF NOTATIONS

Notation	Description
PK	System public key
$\mathbb{UA}$	Universal attributes
PK_a^i	Attribute a's public key at TS_i
sk_a^i	Attribute a's private key at TS_i
MK	Master key
s	Shared secret key
$\mathcal{A}$	Alice's attributes
$\mathcal{T}$	Effective time of Alice's attributes
$\mathbb{A}$	Access control
PK_u	User public key
SK_u	User identity secret key
$SK_{u,a}^i$	User attribute secret key with attribute a and time TS_i

III 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 *Ki* in the sequence, it is computationally infeasible to compute keys *{Kl}* for $l > i$ without having the master secret key, but it is easy to compute keys *{Kj}* 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 $ed \equiv 1 \bmod (p-1)(q-1)$. 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 as $Kctr{+}1 = Kd\ ctr \bmod N$. Authorized users can recursively generate older versions of the current key as $Kctr{-}1 = Kectr \bmod N$ (*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.

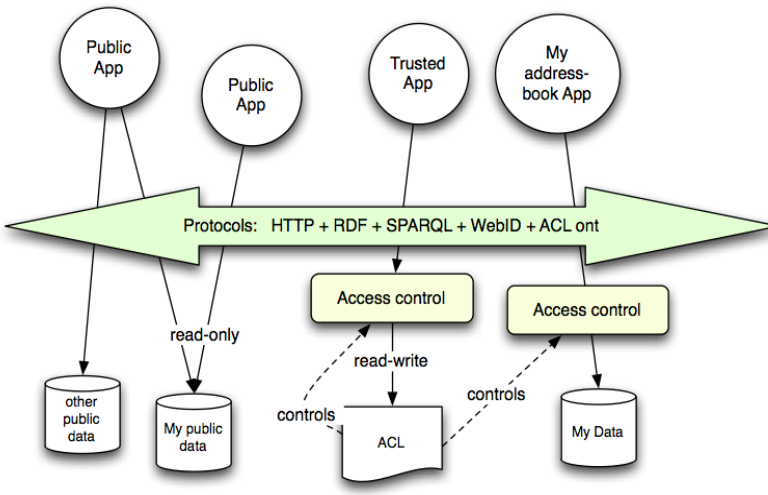


Fig. 1 Cloud computing data storage system model.

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.

We make use of this simple hierarchy to organize data blocks from multiple CSP services into a large size file by shading their differences among these cloud storage systems. For example, in Figure 2 the resources in Express Layer are split and stored into three CSPs, that are indicated by different colors, in Service Layer. In turn, each CSP fragments and stores the assigned data into the storage servers in Storage Layer. We also make use of colors to distinguish different CSPs. Moreover, we follow the logical order of the data blocks to organize the Storage Layer.

/ 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 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 $\langle \mathsf{BI}, \text{TEntry}_{\mathsf{BI}}, j, null, null, \texttt{RevFlag}, \tilde{\bar{b}}\rangle$ to both the CSP and the TTP (OSMR transmission)

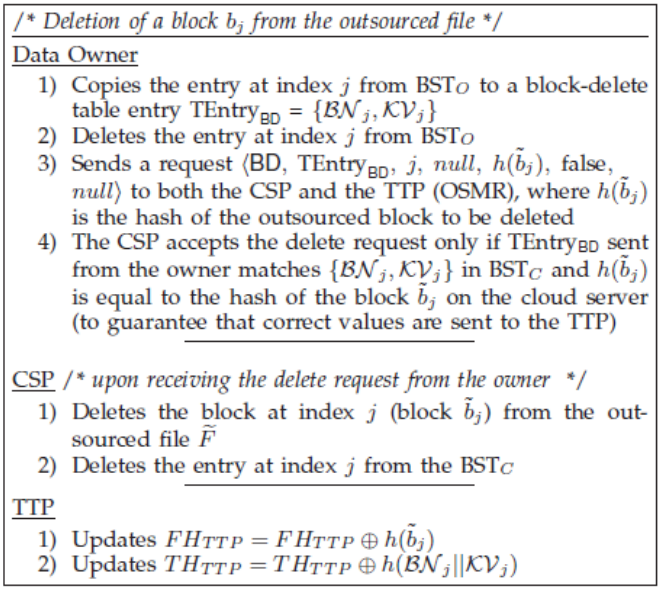
CSP */* upon receiving the insert request from the owner */*

1) Inserts the block $\tilde{\bar{b}}$ after index j in the outsourced file $\tilde{F}$
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 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 BST_O to a block-delete table entry $TEntry_{BD} = \{\mathcal{BN}_j, \mathcal{KV}_j\}$
2) Deletes the entry at index j from BST_O
3) Sends a request $\langle BD, TEntry_{BD}, j, null, h(\tilde{b}_j), false, null\rangle$ 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 $TEntry_{BD}$ 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 $\tilde{F}$
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.

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.

IV CONCLUSIONS

We articulate performance optimization mechanisms for our scheme, and in particular present an efficient method for selecting optimal parameter values to minimize the computation costs of clients and storage service providers. Our experiments show that our solution introduces lower computation and communication overheads in comparison with non-cooperative approaches.

V EVALUATION & CONCLUSION

In this paper, we have proposed a cloud-based storagescheme 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.

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. Seb´e, 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. aData 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] Erway, A. K¨upc¸ ¨ u, 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] 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.