A Provable Multi Store Data Possession in Multi-Cloud Computing Systems Using MVT

Khadarunnisa, Master of Engineering, Sankethika Vidhya Parishad Engineering College, Visakhapatnam & India,
NRGK Prasad, M.Tech, (PhD), Assistant Professor, Sankethika Vidhya Parishad Engineering College, Visakhapatnam & India,

***Abstract*:** Most of the organisations are opting for outsourcing data to cloud service providers (CSP's). Storing and retrieving unlimited amount of data by paying fees metered in gigabytes/month. Most of the customers want data to be replicated on multiple servers across multiple data centres for high levels of scalability, availability and durability of data. In this paper we proposed a map-based provable multi copy dynamic data possession scheme which has following features: It provides evidence to the customers that the Cloud Service Provider is storing as many copies as customer requires. It supports block level operations like block modification, deletion, Insertion and append. And it allows authorized user to access the file copies stored by CSP.

***Keywords*- Cloud computing data replication, outsourcing data storage, dynamic environment**.

I. INTRODUCTION

Outsourcing data to a remote CSP allow organisations to store more data on cloud servers than on private computer systems. Once the data has been sent to a remote CSP which may be untrustworthy, the data owners lose their control over the data. Many researchers have focused on the problem of provable data possession (PDP) and different schemes to audit the data storage. In a typical PDP model, the data owners generates some metadata for some data files further used for verification process through challenge-response protocol.

We propose a map-based provable multi-store dynamic data possession (MB-PMDDP) scheme. This scheme provides guarantee that CSP stores data copies agreed upon the contract. It supports block level operations such as block modification, insertion, deletion and append. We show security of our scheme against colluding servers, and discussing a slight modification on our proposed scheme.

II. PROPOSED SCHEME

The cloud computing used in this work considered in this work consists of three main components (i) a data owner is originally possessing sensitive data to be stored in the cloud. (ii) CSP stores data file of the owner in paid storage space of organisation. (iii) Authorized users can only access the remote data.

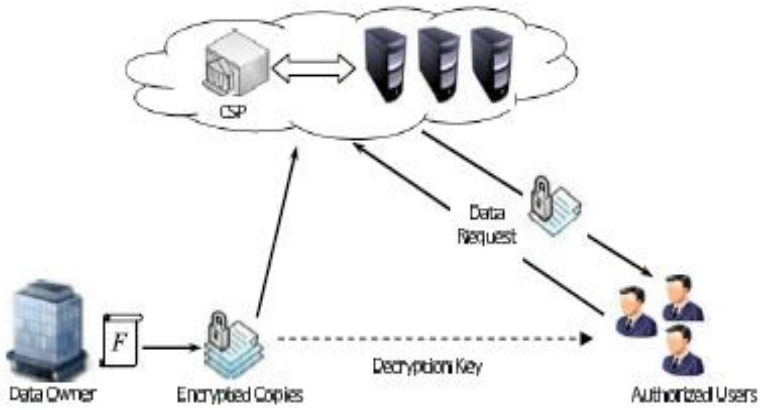

Figure 1. Multi storage in cloud computing model

The proposed scheme consists of seven polynomial time algorithms:

a) **KeyGen():** This algorithm is run by data owner to generate public key (pk) and private key (sk). Where the sk kept secret and pk is publicly known. It return (pk, sk)

b) CopyGen(CN, F): This also runs by data owner. It takes copy number CN as input with a file F, and generates n copies. The owner sends n copies to CSP to store on cloud. It returns n copies of files F'**.**

c) **TagGen**(sk,F'): This algorithm run by data owner to generate output tags/authenticators set Ω by taking private key sk and file copies F'.The data owner sends Ω to the CSP to be stored along with file copies F'.

d) **PrepareUpdate**(D, updateInfo): Data owner will run this algorithm to update the outsourced file copies stored by the CSP. It takes Meta data D and UpdateInfo about the dynamic operation to be performed on a specific block. It returns $D^{/}$ which is a modified metadata and UpdateReq. This request may contain a modified version of previously stored block, new block to be inserted, delete command to delete a specific block from the copies.

e) **ExecUpdate**($F^{/}$, **Ω**, UpdateReq): This algorithm will run by the CSP. File copies F/, the tag set **Ω**, and the request UpdateReq as parameters to the algorithm. And it returns update version of the file copies $F^{//}$ along with updated tag set $\Omega^{/}$.

f) **Prove** ($F^{/}$, **Ω**, chal): this is also run by CSP as proof that CSP is storing n copies of the file. It takes input file copies $F^{/}$, tag set **Ω**, and a challenge chal (sent from a verifier).And it returns a proof β to the data owner as assurance.

g) **Verify** (pk, β, D): This algorithm rum by the verifier or end user. It takes public key pk, the proof β returned from CSP, and the recent metadata D. It returns 1, if the integrity of all file copies is correctly verified or 0 in other case.

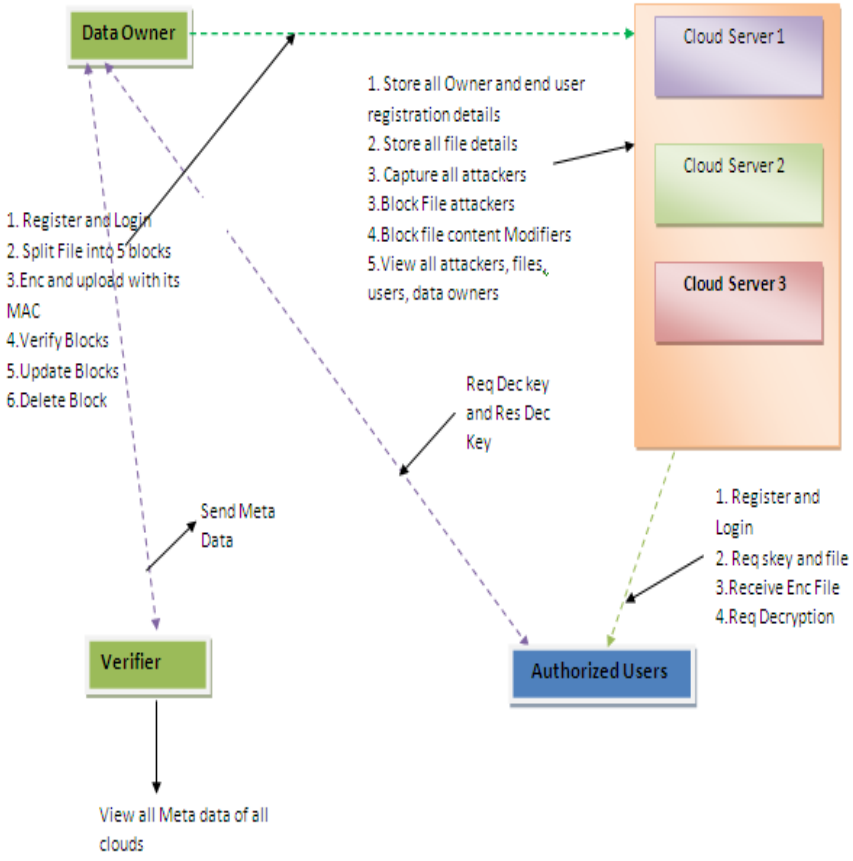

Figure 2. System Architecture

III. IMPLEMENTATION

In proposed MB-PMDDP scheme allows the data owners to update and scale the block of file copies outsourced to cloud severs. The verifier should aware of the new added or modified block of file data by the CSP. This scheme uses a small data structure (metadata), called as map-version table (MVT).

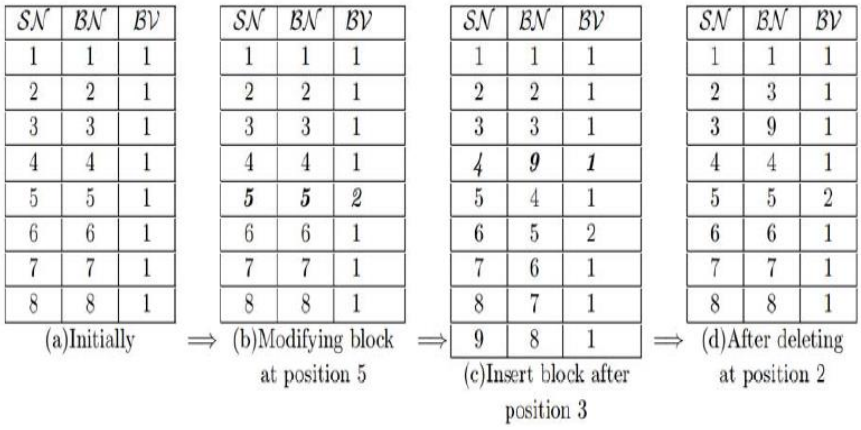

Figure 3. Different operations on MVT

A MVT is a small dynamic data structure stored in verifier side to validate the integrity and consistency of all file copies outsourced to the CSP. MVT has three columns: serial number (SN), block number (BN) and block version (BV).SN is indexing to the file block physical position. BN is counter used to make a logical indexing to the file block. Whereas the relation between SN and BN is like mapping between logical number and physical position of the data file. And BV indicates the current version of the file block. For initially created data file BV=1, if specific block is updated BV incremented by 1.

a) Notations:

- F is data file to be outsourced, and is composed of a sequence of m blocks. $F=\{b_1,b_2...b_m\}$.
- $¥_{key\ (.)}$ is a pseudo random permutation (PRP):key * $\{0,1\}^{log2(m)}$ ->$\{0,1\}^{log2(m).1}$
- $€_{key}$ (.) is a pseudo random function PRF: key * {0,1}* ->Zp (P is a large prime.
- Bilinear Map/pairing: Let G_1, G2 and G_T be cyclic groups of prime order p. Let g1 and g2 be generators of G1, and G2, respectively. A bilinear pairing is a map
- H(.) is a map-to-point hash function :{0,1}*->G_1.
- E_K is an encryption algorithm with strong diffusion property. e.g., AES.

b) MB-PMDDP procedural steps:

- Key Generation: Let e: G_1*G_2->G_T be a bilinear map and 'g' a generator of G_2. Data owner will run the KeyGen to generate private key x € Z_p and public key $y=g^x$ € G_2.

- Generation of distinct copies: Data owner run CopyGen algorithm to create distinct copies F/**.** The block bij is generated by concatenating a copy number i with the block bj, then encrypting using an encryption scheme E_k, i.e., bij = Ek(i||bj). The number of block sectors *s* depends on the block size and the prime *p*, where *s* = |block size|/| *p*| (|.| is the bit length).

  The authorized users need only to keep a single secret key *K*. Later, when an authorized user receives a file copy from the CSP, he decrypts the copy blocks, removes the copy index from the blocks header, and then recombines the decrypted blocks to reconstruct the plain form of the received file copy.

- Generation of Tags: Each file *F* comprising the file name, the number of copies for this file, and the random values $\{u_k\}_{1\le k\le s}$. Note that if the data owner decides to use different block size (or different *p*) for his different files, the number of block sectors *s* and hence $\{u_k\}_{1\le k\le s}$ will change. We assume that $I\,D_F$ is signed with some owner's signing secret key (different than *x*), and the CSP verifies this signature during different scheme operations to validate the owner's identity. The MVT is stored on the local storage of the owner (or any trusted verifier).

- Dynamic operations on data copies: The dynamic operations are performed at the block level via a request in the general form $I\,D_F$, *BlockOp, j,* $\{b_i^*\}_{1\le i\le n}$, σ_j^* , where $I\,D_F$ is the file identifier and *BlockOp* corresponds to block modification (denoted by BM), block insertion (BI), or block deletion (BD).The parameter *j* indicates the index of the block to be updated, $\{b_i^*\}_{1\le i\le n}$ are the new block values for all copies, and is the new aggregated tag for the new blocks.
  - Modification: For a file F, suppose the owner wants to modify a block bj with bj/ for all file copies F/.The owner runs the PrepareUpdate algorithm to do the following:
    1. Update $BV_{j=}BV_j+1$ in the MVT.
    2. Create n distinct blocks where b_{ij} is fragmented.
    3. Creates a new tag set.
    4. Sends a modified request to the CSP.
  - Insertion: To perform the insertion of a new block b after position j in all file copies, the owner runs the PrepareUpdate algorithm to do the following:
    1. Construct a new table entry, and insert these entries in the MVT after position j.
    2. Create n distinct blocks, where bi is fragmented into s sectors.
    3. Create a new tag for each block, then generate an aggregate tag. Note that $BN_{j+1}$is logical number of the new block with current version.

4. Send an insert request to the CSP.

- ❖ 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.
- ❖ Delete: When one block is deleted all sub sequent locks are moved one step forward. To delete a specific data block at position j from all copies, the owner deletes the entry at position j from the MVT and sends the delete request to the CSP. Upon receiving this request, the CSP runs the ExecUpdate algorithm to do the following:
  1. Delete the blocks, and output a new version of the file copies.
  2. Deletes σ_j from Φ and outputs $\Phi^{/}=\{\sigma_1,\sigma_2.....\sigma_{m-1}\}$.

- ➢ Challenge: For challenging the CSP and validating the integrity and consistency of all copies, the verifier sends c and two fresh keys at each challenge :a PRP(Π) key k_1 and a PRF(Ψ) and key k_2. Both the verifier and CSP use Π keyed with k_1 and the Ψ keyed with k_2 to generate a set Q. The set of random indices {j} is the physical positions of the blocks to be challenged.

- ➢ Response: The CSP runs the Prove algorithm to generate a set Q of random indices and values, and provide an evidence that the CSP is still correctly possessing the n copies in an updated and consistent state.

- ➢ Verify Response: Upon receiving the proof from the CSP, the verifier runs the Verify algorithm to check the verification equation. The verifier uses the set of random indices {j} and the MVT to get the logical block number BN_j and the block version BV_j of each block being challenged. If the verification equation passes, the Verify algorithm returns 1, otherwise returns 0.

$$\left(\left[\prod_{j,r_j\in Q} H\left(ID_F \,||BN_j||BV_j\right)^{r_j}\right]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y\right) \quad \text{..... (1)}$$

Once the attempt to slightly modify the MB-PMDDP scheme to reduce the communication overhead by a factor of n via allowing the CSP to compute and send μ. However this modification enables the CSP to simply cheat the Verifier.

The proposed MB-PMDDP scheme supports public verifiability where anyone, who knows the owners public key but is not necessarily the data owner, can send a challenge vector to the CSP and verify the response. Public verifiability can solve disputes that may occur between the data owner and the CSP regarding data integrity. If such a dispute occurs, a trusted third party auditor (TPA) can determine whether the data integrity is maintained or not. Since the owner's public key is only needed to perform the verification step, the owner is not required to reveal his secret key to the TPA.

Verifier **CSP**

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

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

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

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

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

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

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

Figure 4. Challenge response protocol in the MB-PMDDP scheme

c) Implementation:

Implementation of the present scheme consists of three modules: OModule (Owner module), CModule(CSP Module), and VModule(Verifier Module), which runs on the owner side, is a library that includes KeyGen, CopyGen, TagGen and PrepareUpdate algorithms. CModule is library that runs on Amazon EC2 and includes ExecuteUpdate and Prove algorithms. VModule is a library to be run at the verifier side and includes the Verify algorithm.

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

Implementation settings: A "Large" Amazon EC2 instance is used to run CModule. Through this instance, a customer's gets total memory of size 7.5GB and 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute units each). One EC2 compute unit provides the equivalent CPU capacity of a 1.0-1.2GHz 2007 opteran). The OModule and VModule are executed on a desktop computer with 2GHz processor and 3GB RAM running Windows 7. We outsource copies of a data file of size 64MB to Amazon s3.Algorithms (encryption, pairing, hashing etc) are implemented using MIRACL library.

IV. IDENTIFYING CORRUPTED COPIES

MB-PMDDP scheme identify the indices of corrupted copies. The proof P generated by the CSP will be valid and will pass the verification only if all copies are intact and consistent. If there are one or more corrupted copies then the auditing procedure fails. To handle this situation a slightly modified version of MB-PMDDP scheme can be used where owner generates tag σ_{ij} for each block b_{ij}, but does not aggregate the tags for the blocks at the same indices in each copy.

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

The BS (binary search) algorithm takes four parameters: σ list, μ list, Start that indicates the Start index of currently working list and End to indicate the Last index of this list. The initial stage BS algorithm takes (σ list, μ list, 1, n). This slightly modification to identify the corrupted copies will be associated with some extra storage overhead on the cloud servers, where the CSP has to store mn tags for the file copies $F^{/}$ (m tags in the original version).

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

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

Figure5- Procedural steps of indentifying the indices of corrupted copies

CSP to remain in business and maintain a good reputation, invalid responses to verifier's challenges are sent in very rare situations and thus the original version of the proposed scheme is used in the most of the time rather than the modified one.

V. CONCLUSION

Outsourcing data to remote servers has become a growing trend for many organisations to alleviate the burden of local data storage and maintenance. We faced a problem where the multiple copies of dynamic data are stored on un trusted cloud servers.

The PDP scheme in this paper supports outsourcing of multi-copy dynamic data, where the data owner is capable of not only achieving and accessing the data copies stored by the CSP, but also updating and scaling these copies on the remote servers. This is the first scheme which addresses multiple copies of dynamic data. Moreover, the proposed scheme supports public verifiability, enables arbitrary number of auditing, and allows possession-free verification where the verifier has the ability to verify the data integrity even though he neither possesses nor retrieves the file blocks from the server.

VI. REFERENCES

1. C.E. Shanon, "Communication theory of secrecy systems", Bell syst Tech. J., vol.28
2. R.C.Merkle,"Protocols for public key cryptosystems." In Proc IEEE Symp. Secur. Privacy.
3. Z.Hao.S.Zhong, and N.Yu,"A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability". IEEE Trans.Knowl. Data Eng. Vol 23.
4. R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in Proc. 28th IEEE ICDCS, Jun. 2008
5. Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in Proc. 2nd Int. Symp. Data, Privacy, E-Commerce
6. K .D. Bowers, A. Juels, and A. Oprea, "Proofs of irretrievability: Theory and implementation," in Proc. ACM Workshop Cloud Comput. Secur
7. D. L. G. Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer," IACR (Inter-national Association for Cryptologic Research) ePrint Archive, Tech.

8. F. Sebé, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," IEEE Trans. Knowl. Data Eng., vol. 20, no. 8, pp. 1034–1038, Aug. 2008.
9. Amazon Elastic Compute Cloud (Amazon EC2),{online} . Available: https://aws.amazon.com/ec2/, accessed Dec 2018.
10. Amazon Simple Storage Service (Amazon S3),{online} . Available: https://aws.amazon.com/s3/, accessed Dec 2018.
11. Amazon EC2 Instance Types (Amazon EC2),{online} . Available: https://aws.amazon.com/ec2/, accessed Dec 2018.
12. A.F. Barsoum and M.A. Hasan (2011)."On Verifying dynamic multi data copies over cloud server", IACR Cryptology ePrint Archive, Tech. Available : http://eprint.iacr.org/