International Journal of Innovative Research in Computer Science and Technology (IJIRCST)
ISSN (Online): 2347-5552, Volume-13, Issue-4, July 2025
DOI: https:/doi.org/10.55524/ijircst.2025.13.4.8
Article ID IRP-1662, Pages 76-84
www.ijircst.org

Towards a JAR Based PDP Data Integrity Assurance in Cloud Computing

Rizwan Saleem[1], Nazia Akram [2], and Sumaira Ameer Jan [3]

[1] Visiting Lecturer, Department of Computer Science, Islamia University, Bahawalpur, Pakistan
[2, 3] Masters Student, Computer Science and Technology, Dalian Maritime University, China

Correspondence should be addressed to Rizwan Saleem; Rizwanmcs09@gmail.com

Received 27 June 2025; Revised 12 July 2025; Accepted 25 July 2025

ABSTRACT- Cloud storage as a service provides scalability and high availability as per the user's need, without considerable investment in infrastructure. However, data security risks, like confidentiality, privacy, and integrity of the outsourced data, are associated with this model. Over the years, techniques like remote data checking (RDC), data integrity checking or data integrity protection (DIP), provable data possession (PDP), Proof of Storage (POS) and Proof of irretrievability (POR), etc., have been devised to frequently and securely check the integrity of outsourced data. Cloud storage service is always assumed to be unreliable and insecure, so a secure and efficient data integrity checking mechanism is of utmost importance. This thesis focuses on making the existing PDP scheme more efficient in computation, storage, and communication cost for extensive data archives. By utilizing JAR and ZIP technology capabilities, we have reduced the cost of searching the metadata in the proof generation process from O(n) to O(1). Due to direct access to metadata, disk I/O cost is reduced, and we achieved 50 to 60 times faster proof generation for large datasets. Our proposed scheme achieved a 50% reduction in the storage size of data and respective metadata, providing storage and communication efficiency.

KEYWORDS: Cloud Computing, Integrity Verification, Cryptographic Techniques, Provable Data Positioning

I.INTRODUCTION

Cloud computing delivers a range of services that provide flexibility, multi-tenancy, agility, and high availability to meet user requirements, all without the burden of heavy infrastructure investments [1]. Small-scale organizations with large data storage needs often lack the resources to manage their own data centers effectively. Managing vast volumes of archival data, such as extensive tape backups, becomes challenging, despite the relative ease of maintaining such collections[2].

Software as a Service (SaaS) in the cloud enables users to benefit from its elastic nature. However, when clients outsource their data to the cloud, they lose direct control over it, raising concerns about the authenticity and integrity of their stored information. Detecting data corruption during normal access is difficult, and recovery may be impossible if the issue is discovered too late[3]. Archival data, although rarely accessed, is highly valuable. Retrieving and transmitting large archival files also incurs high I/O costs, which limits the scalability of SaaS solutions when full data integrity checks are required. Furthermore, cloud storage providers (CSPs) may have conflicting interests, including financial incentives, potential dishonesty, or even breaches of data confidentiality[4]. While CSPs are generally bound by service-level agreements to protect client data, users cannot fully trust CSPs alone to maintain data security. SaaS models face security challenges related to data confidentiality, privacy, and integrity despite their usability benefits. Therefore, SaaS platforms must implement reliable mechanisms to guarantee the integrity of client data in all circumstances[5]. It assists dynamic operations similar to block modification, append insertion and deletion. It also secures our system[6]. Wang et al.[7] planned enhanced POR supports (MHT) Merkle Hash Tree. Major characteristics integrated public verifiability by 3rd party auditors and dynamic functions. Later, in [8], PDP was planned. They used authenticator Homomorphic in nature with casual masking and were intelligent and sharp enough to do public verifiability. Afterward, they handle the difficulty of O(n) for confirmation by TPA with the help of BAL. In Remote Data Checking for Network Coding-based Distributed Storage systems. They used same construction of a message authentication code, proposed in [6] by a combination of universal hashing with a PRFs It increased communication overhead with reduced storage overhead [9].

Classification of data integrity schemes

There are two categories for data integrity checking schemes

- Provable Data Possession(PDP)
- Proof of retrieval

A. Provable Data Possession (PDP)

PDP schemes Remote Data Checking Using Provable Data Possession [10] are more probabilistic as these schemes use sampling or examining random chunks as an alternative to reading entire files for verification. In PDP, novel data is initialized to produce various metadata, which is located with novel data. Afterward, this metadata is used to confirm the integrity of user's data stored on the cloud. These systems can only recognize the dishonesty in data except not hold up corrupted or dishonest data recovery.

B. Proof of Retrievability (POR)

Proof of retrievability mechanisms is primarily comparable to PDP; however, they also endow the data recovery. POR method uses unneeded or redundant programming, and encoding data, therefore, offers the recovery if any failure happens. Provable data possession [11] scheme may be changed to Proof of retrievability via fault correcting or removal codes. Inspection protocol of Proof of retrievability offers the assurance that Cloud Service Provider holds all data and that it is recoverable, but Provable data possession makes sure that server has the significant part of data. Means PDP cannot recognize smaller data corruptions (1KB) data loss in (20GB) files if any deterioration is recognized afterward. Provable data possession is not responsible for the recovery.

In this Paper, the Provable data possession scheme is selected. Since it is for static data, our focus is on the data archival in static nature and never usually accessed. The planned scheme's potency is examined to offer 99% possession assurance by using 4.6% of chunks of the whole file. This scheme is suitable for all kinds of data as it is arrangement data independent.

C. Existing PDP Schemes

Existing PDP meetup the basic 6 out of 7 prerequisites. But it is unable to fulfil the 7th requirement that ensures the design is authentic. This design is for massive datasets. The essential degrading factors for this scheme are metadata per chunk, the size of chunks chosen, and several other factors. Now, based on this, we planned an approach that reduces the effects of these parameters. Our designed system is far better than the older one and more effective in cost, Time, and performance. These things also bound, how much time data holder can confirm the unity of his data it's relatively difficult to approach with a strategy that can hold all these presentation corrupting components in a mode these presentation does not affect. Well, management of these all factors can create existing schemes more than competent. The basic idea is to point the chunk and relevant metadata jointly in a solitary zipped file in our proposed formulation. In a similar manner syndicate all chunk's zip file forename, e.g., for chunk at index 1, zip file forename is "z1.zip" so with no any calculation we see the pathway of to each one chunk equivalent to exacting index and therefore we have straight entrance to some chunk data and personal metadata. This scheme ready-made the evidence generation of available schemes about 40%-60% quicker. Another advantage of the zip scheme is cheaper.

Table 1: Overall view of Data Integrity Checking Schemes Comparison

Scheme	**Technique(s) Used**	**Limitations**
An improved dynamic provable data possession model (2011)[12]	Used skip list and hashes	The client needs to store some secret values thus need little extra storage like for 4GB file needs 2MB storage
A Position Paper on Data Sovereignty: The Importance of Geo locating Data in the Cloud (2011)[13]	MAC-based PDP with network delay measurement capabilities	Cannot guarantee that additional copies of data are not instantiated outside of a prescribed Geographic area.
Fair and Dynamic Proofs of Retrievability(2011) [14]	Used authenticated data structure (range-based 2-3 trees) and incremental signature scheme and ECC (Error-correcting codes)	Low performance because of Error-correcting codes
Robust Dynamic Provable Data Possession (2012)[15]	Reed-Solomon (RS) codes based on Cauchy matrices	
Efficient, dynamic and identity-based Remote Data Integrity Checking for multiple replicas (2019)[16]	a novel identity-based RDIC scheme, namely Efficient, Dynamic and Identity-based Multiple Replication Provable Data Possession (EDID-MRPDP) without the burden of PKI	
Proof of Possession for Cloud Storage via Lagrangian Interpolation Techniques(2012) [17]	Secure pseudorandom numbers (SPRN), Lagrangian Interpolation	
Provable Data Possession Using Sigma-protocols (2012) [18]	Signatures based on Okamoto protocol using pseudorandom permutations and pseudorandom generators	No support for dynamic data operations. Limitedtostaticdata only
Ensuring distributed accountability for data sharing in the cloud (2012) [19]		
Knox: Privacy-preserving Auditing for Shared Data with Large Groups in the Cloud (2012) [20]**Error! Reference source not found.**	Used group signatures for homomorphic authenticators to provide public auditing. Rely on Homomorphic MACs for storage efficiency. Index hash tables as identifiers of chunks	Unable to identify minor corruptions with sampling. Computation cost increases with the increase of sampled chunks toidentify small no of corrupt chunks
Survey on cloud data integrity proof techniques (2012)[21]	Rank-based authenticated skip lists	High communicatin cost and not secure
Secure and Efficient Proof of Storage with Deduplication (2012) [22]	Merkle Hash Tree, Pseudorandom functions, Hashing	Less efficient and less secure than existing POW schemes. But provides public auditing where existing POW schemes do not.

II. PROPOSED SCHEME

We have designed the Provable Data Possession method, and our scheme resolves the problem by using the link list and array. We can quickly contact any index in arrays, and in the link list, we have to navigate it to attain the desired index. So, the indexes must be in sorted order in the link list.

Figure 1: Stream formed PDP

We did not require to navigate the linked list repeatedly as we build the subfile metadata tags, chunks, and index in a single file. We recognize the path of every chunk to relax, and there is no computation, and that's why we have access to both the chunk and the personal metadata. Programmable JAR offers the stream of our formed PDP method derived from the control logic jointly. Figure 1 display the stream of our formed PDP method derived from the PDP design of Ateniese et al.[1].

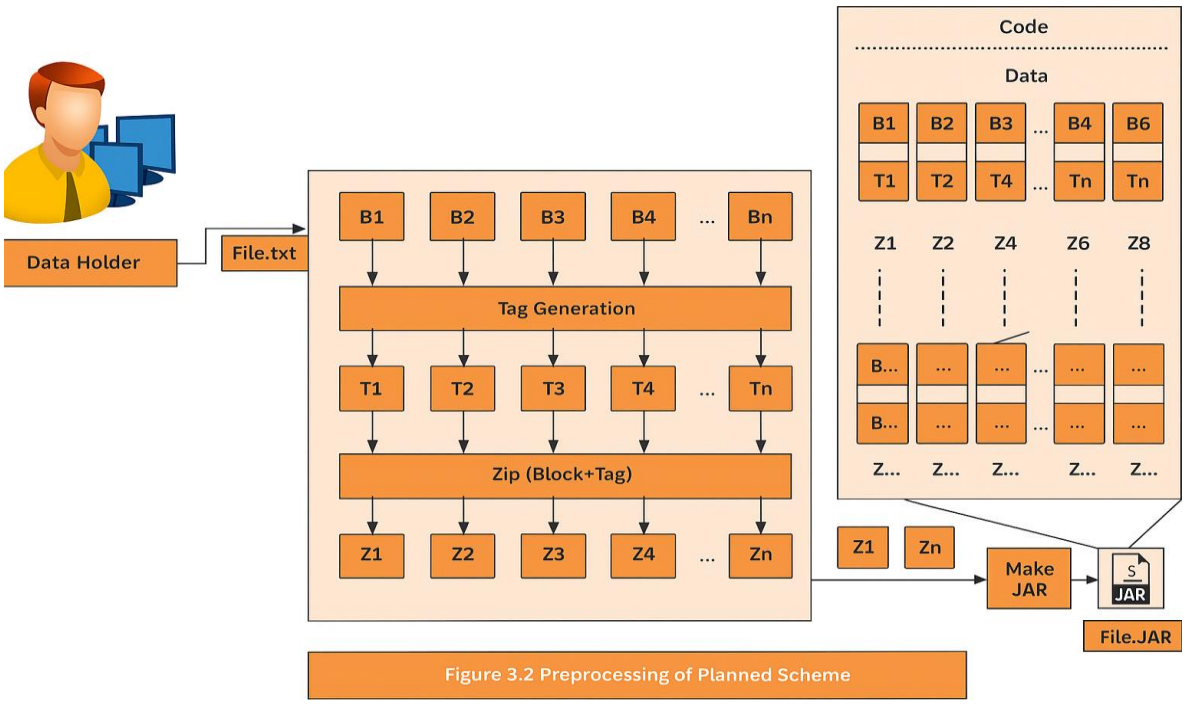

Figure 2: JAR file is transferred to a server

Phase 1: The first step is to process the input file, exchange it into an executable JAR, and then transfer it to the server. It is further elaborated in Figure 2 Phase 2:The data holder establishes the integrity once the JAR file is transferred to a server. The data holder produces the challenge. The server has Proof against this challenged, and then the data holder confirms the integrity of his data. In the figure, the prepossessing is explained. The input file has a fixed length symbolized by Bi in Bi I means to index, and for every chunk a flag is generated, and Ti represents that. The Ti and Bi are jointly appended in a single zip file, and then the zip is added into the JAR. The whole procedure is repeated for all chunks of the input file. Afterward, JAR has collected the executable code work that helps to assure proof production. Figure 3 shows the complete work and how our designed approach works

Step 1: A JAR file is created by the data holder in which data is arranged into fixed lengths, and flags are generated for every chunk. Then the flag and the chunk are jointly zipped, and then the zip is added into the JAR.

Step 2: Once JAR is created, the related code is also added that assist in accessing the data and the metadata with the help of proof generation on the server.

Step 3: Transferring JAR to the server, which contains the chunks metadata and the related code

Step 4: The user checks the integrity of his data by sending sampled chunk to the server after data is located over the cloud.

Step 5: Proof generation code is called up once the server bypasses the challenged chunk. The JAR file does the mining process because the chunks and metadata flag are in the JAR file.

Step 6: After all the process is done, the last Proof returns to the storage server by JAR.

Step 7: The proof that is created by the JAR is sending back to the data holder for verification.

Step 8: Data holder verifies the Proof of possession presented by storage space server via the similar verification technique of Ateniese PDP planned. If any chunk is modified, deleted, or an extra piece is added afterward, verification will fall short; otherwise, it will succeed.

Data holders confirm the verification via the same technique as Ateniese planned. If any change occurs, then Prooffalls shorts. Other than that, it will succeed.

Step 9: The result shows to the data holder.

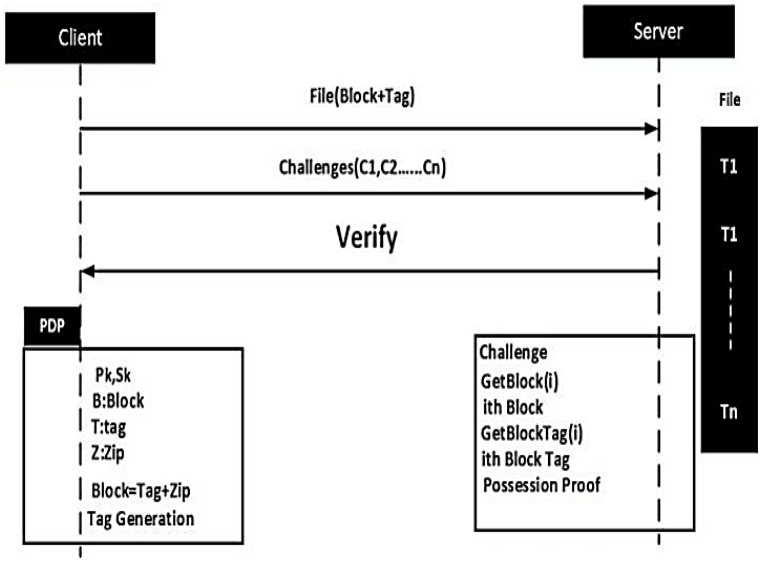

Figure 3: Our designed approach works

In figure 4 shows that the client sends the file containing blocks and the metadata tags to the server and the challenges are also sent by the client-side to the server to verify the integrity. In return, the server accepts the client's challenges, gets the block into the file, takes the tags contained in it, and returns the Proof of possession to the client.

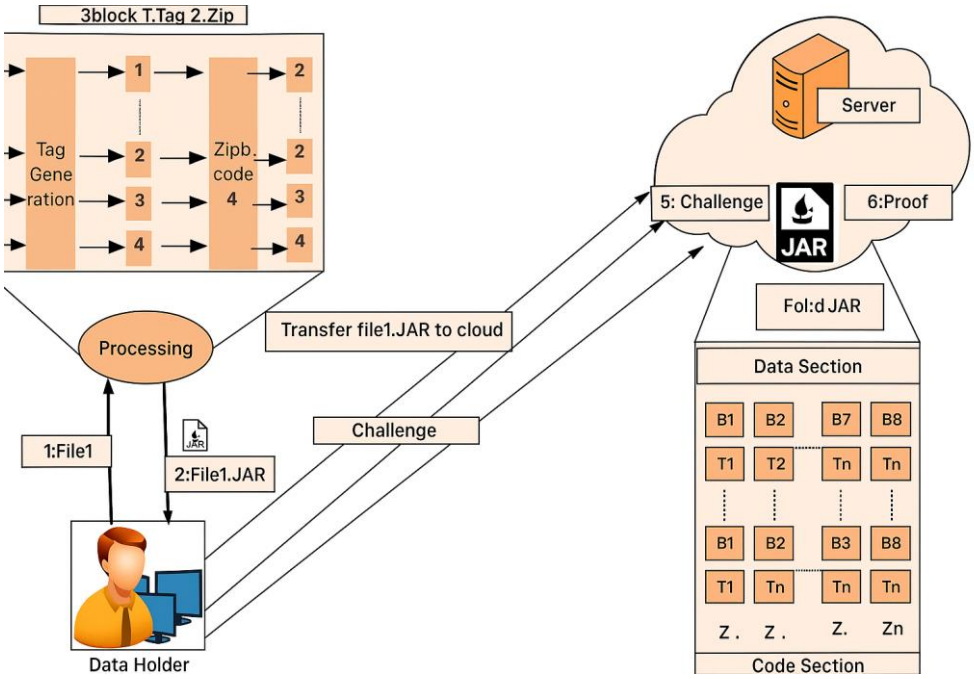

Figure 4: client sends the file containing blocks and the metadata tags

III. RESULTS AND DISCUSSIONS

Algorithms of Proposed Method

GenKey(1k): .

Our approach is based on the following algorithms GenrateKey(OneK): (CK, xk) is a key generation algorithm that executes on the client-side. The working of the PDP relies on these keys. These arguments are like security parameters S, and it returns a combination of public and secret keys(CK, xk).

Preprocessing File(file): JAR treats the input file as a collection of chunks and generates the homomorphic flags for all chunks. Each chunk and a respective flag is zipped in a single file and added to JAR. At the end of

preprocessing, reference to the JAR file, i.e., JAR, is returned.

Generate Proof(JAR,C): V is executed on the server at the client's request. Its arguments are JAR file as an input holding ordered chunks and flags, a challenged C. it generates the Proof of possession V for the challenged chunk by accessing chunk and flag inside the JAR.

VerifyProof(V,C): {"true, " false"} is executed by the data owner to verify the Proof provided by the server. Arguments of it are challenge C and Proof of possession V. if verification of Proof is successful means that the server provided a valid proof for the challenged chunk, then it returns true, and if Proof is invalid, then it will return false, indicating the misbehavior of the server

Making JAR from the Input

```
1:  procedurePREPROCESSFILE(file)        >Input data file to beprocessed
2:  CHUNK SIZE=4096                        >4KB
3:  offset =0                            >bytes offset in file
4:  i =0                                 >index of chunk infile
5:  jar = CREATEJAR()
6:  Bi = READFILE(file, offset, CHUNKSIZE)
7:  whileBi=/                            EOF do d End of file not reached
8:  Ti = EXISTINGPDP.FlagChunk(Bi,i)
9:  Zi = MAKEZIP(Bi,Ti)
10: jar.ADDTOJAR(Zi)
11: offset = offset + CHUNK SIZE
12: i = i +1
13: Bi = READFILE(file, offset, CHUNKSIZE)
14: endwhile
15: returnjar
16: end procedure
```

Algorithm 1

Algorithm 1 describes preprocessing step of the proposed PDP scheme shown in Figure 2. It takes an input file and treats it as a collection of 4k bytes chunks. Each chunk (Bi) Is identified by its index (i). Flag (Ti) is generated using the same construction of PDP against each chunk using its index and data bytes. Then both data bytes and flag are zipped together in a zip file (Zi). The name of the zip file is the index of a chunk, e.g., 1000.blk. This cycle is repeated for all chunks of the input file. A JAR file is created having code and data section. The code section contains the necessary code that provides access to data and metadata in the proof generation process, while the data section holds the zip files produced against all file chunks. This approach provides direct access to both data by sending flag for a particular index.

Algorithm 2 Generate Proof Of Possession

```
1:  procedure GENERATEPROOF(jar, C) d c is challenged chunks index collection
2:  i =1d index of challenged chunk infile
3:    p =EXISTINGPDP.CREATEPROOF()
4:  for index=0 to C.size()-1 do
5:    i =C[index]
6:  Bi =jar.GETChunk(i)
7:  Ti =jar.GETChunkFlag(i)
8:  p = EXISTINGPDP.UPDATEPROOF(Bi,Ti)
9:  p = EXISTINGPDP.UPDATEPROOF(Bi,Ti)
10: returnjar
11: end procedure
```

Algorithm 2

Algorithm 2 describes how Proof of possession is produced on a cloud storage space server with the assistance of JAR. A challenge (C) is an unordered collection of sampled chunk indexes, and a jar is a predictable file for which a

storage space server has to offer Proof of possession. The server gets data chunk (Bi) and flag (Ti) against challenged chunk index (i) from JAR and utilizes it for proof generation (using the same method of PDP in [1]). This series is repeated for all chunks indexes precise in a challenge and at the end of Proof of possession is returned. Our planned method does not change the internals of the novel PDP method proposed in [1]. The only transformation is that it bases on JAR to present data chunks and flag for the specific challenged index as an alternative of reading itself. Since every chunk and its flag are zipped in solitary file and its forename is index of chunk, so is directly available without any additional computation. This advance lessens disk access mainly and thus amplifies performance immensely.

Analysis of Storage and communication

For transferring data on the cloud, you have to pay for both storage and bandwidth. That's why it matters—data size along with meta data effects on transfer time. We have evaluated for both that include data and respective tags. We have increased the file size gradually and performed tests. The performance performing parameters are

Data block size = 4KB

Metadata tag size = changeable length

The comparison shows that the designed scheme achieved30 a % reduction in size. It is effective in communication and storage too. Bandwidth utilization and communication time are lesser in data transfer. There is same input data block. The reduced size is not because of the meta data size the reason is because we are using the ZIP technology. In old PDP, all the data and metadata tags are uncompressed files, and here we have placed all the data and the metadata tags in a zip file. That is why the output size is Time analysis of verification process also reduced.

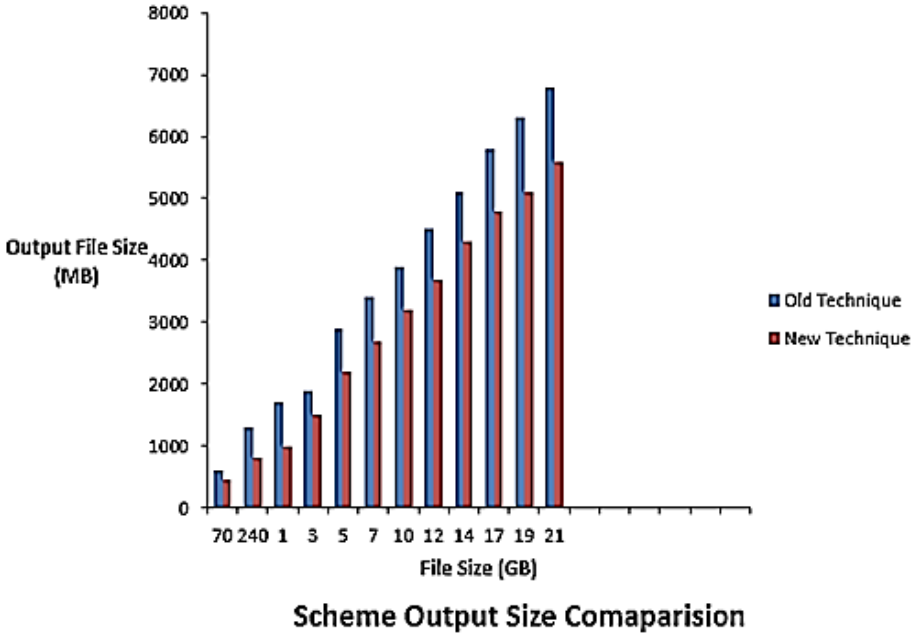

Figure 5: Size comparison

If the input file size is greater than the communication, storage cost will also increase. Data holder produces a challenge that contains a fixed number of blocks indexes chosen from all blocks. Then cloud storage has to provide proof of possession against indexes. So, server needs to access all these metadata tags that are challenged. The Time taken by the server to calculate Proof of possession for these two old and new schemes is compared for several file sizes. 4.2

Since 4KB is tiny, the file size and number of blocks will also increase, and the Proof of possession time will also increase. We have chosen 460 blocks randomly. The comparison shows that the new scheme performs better as compared to the old method. The performance of the old PDP goes down as the file size increases, and the new process generates 50 to 60% faster Proof of possession. The I/O function is performed for searching the metatags in the old scheme. The performance of the old PDP is limited in all aspects.

Time analysis - Before placing the data over the cloud, the metadata tags are generated using teleprocssing these tags are later used for Proof of possession verification. Prepossessing is done on client side. Our proposed scheme has different prepossessing. Therefore, we compare the old and new prepossessing. Figure 6 shows the prepossessing.

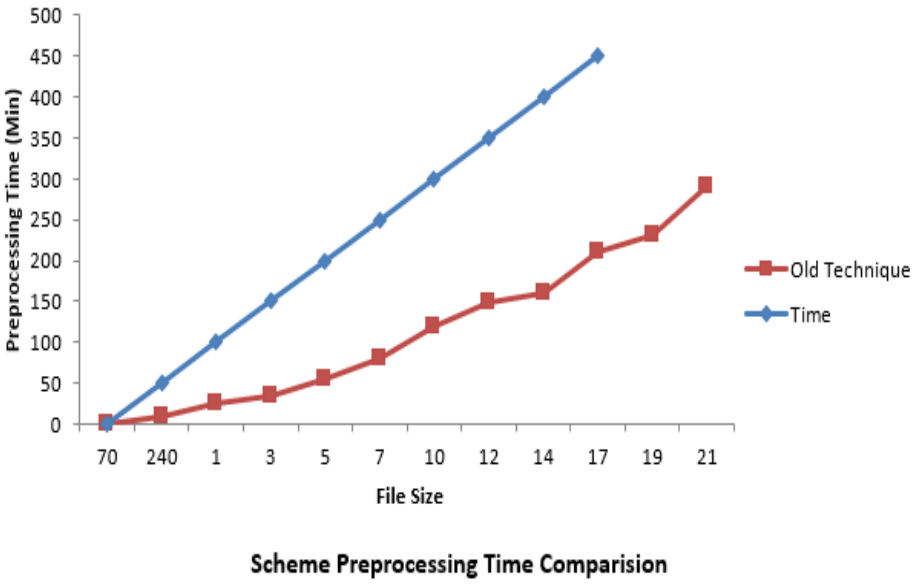

Figure 6: Preprocessing time comparison

Data block size = 4KB
Tag size = Changeable
Prepossessing is done at the client-side, so we have considered a normal machine with specifications with 80GB HDD and two processors, Ubuntu 12.04(OS) with 3GB ram.
Our scheme is efficient and cost-effective but requires prepossessing Time more. 4.5 graph shows computation time difference. But its overhead is only oneTime as compared to the old PDP.
Effect of data block size.
The work of zipping the blocks and tags in the new PDP reduces the size of the file. But when the zip file is inserted in the JAR, as the number of entries reaches above 1000, then the JAR entry takes more Time, and it increases in Time. After the number of entries gets, more the new entries get JAR.
And then the processing time increases due to this as well. These can be improved in two ways.

- If we use a larger block size, then the total number of blocks will decrease, and entries in JAR did not take additional Time for adding in zipping, so it will take lesser teleprocessing.
- We can use a hierarchical directory structure inside JAR for zip files. This means we have multiple JAR directories, and it will take constant time.
- To check prepossessing Time and output, we do tests by keeping the file size the same and increasing the block size.

File Size = 10.7
When we increase the block size, the total number of blocks is reduced and affects the prepossessing and verification Time and output time. A lesser number of blocks means lesser effort is required to generate the meta-tags. If meta-tags size is reduced, it reduces the output time and length. In our proposed scheme, enhanced and better Time is achieved while setting the larger block size. The verification time will also decrease as the number of blocks is lesser. The results are shown in the figure 7.

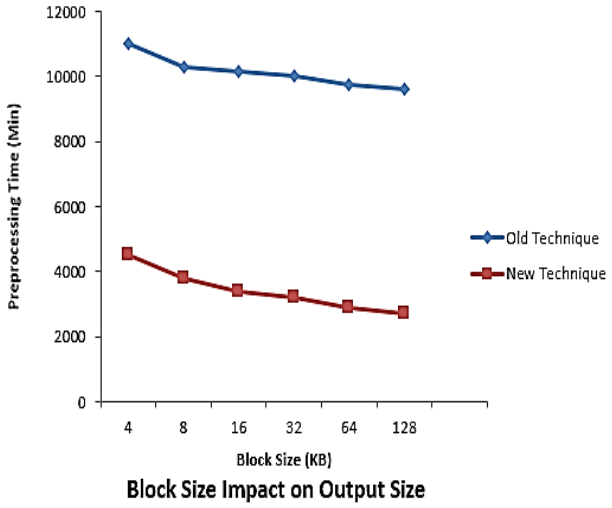

Figure 7: Result

IV. SUMMARY

The efficiency matters the most, and the lesser computation and verification Time, the better approach is Analysing the results that we performed, existing PDP has performance issues, and communication and analysis cost increases as the file size increases. The main reason is the variable length of metadata tags. The other reason is the number of block sizes. This impact can only be better if we increase the block size. If the number of blocks is smaller, the number of file blocks increases, and more computation and searching are required to increase the I/O cost. It will also affect storage and communication cost because more metadata tags are generated for smaller block file sizes that impact the storage over cloud and transfer Time.

In the proposed scheme only, prepossessing time matters, and the prepossessing is done on the client-side, so it is acceptable. The prepossessing time can also be improved using the larger block size. The proposed scheme has efficiency in computation communication storage on the cloud. Different parameters do not degrade the performance of the proposed project because there is no need to search as these metadata tags are directly accessible that improves the block accessible cost and verification. Time cost. As a result, communication and storage cost becomes lesser, and this scheme provides more storage efficiency, and we used JAR technology that allows communication efficiency.

V. CONCLUSION

PDP is the first probabilistic verification method. Homomorphic tags are generated against data blocks that verify Proof. Our scheme relies on a similar structure as Provable Data Possession by Ateniese et al. we achieve parallel scalability using similar tag generation and data integrity. We divide a file into the smallest sub-files and generate homomorphic tags before outsourcing the data. We zipped the tags and the file into a zip file and then transferred it into a JAR, and in the end, when the prepossessing is done, we have a single JAR file that contains code and provides the integrity proof. Afterward, the JAR file is transferred to the cloud instead of the original data file. We achieve the 50% reduction data size, and our results show that and our performance is far better than the old PDP.

ACKNOWLEDGEMENT

I am thankful for dr. Tehmreem Masood and dr. Ahmad Niaz for valuable guidance during this research also acknowledged superior university Lahore, Pakistan for providing the computing resources & references.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] G. Ateniese, r. Burns, r. Curtmola, j. Herring, l. Kissner, z. Peterson, and d. Song, “provable data possession at untrusted stores,” in proc. 14th acm conf. Comput. Commun. Secur. (ccs), new york, ny, usa, 2007, pp. 598–609. Available from: https://doi.org/10.1145/1315245.1315318

[2] G. Ateniese, r. Di pietro, l. V. Mancini, and g. Tsudik, “scalable and efficient provable data possession,” in proc. 4th int. Conf. Security and privacy in commun. Netw. (securecomm), new york, ny, usa, 2008, pp. 9:1–9:10. Available from: https://doi.org/10.1145/1460877.1460889

[3] R. Curtmola, o. Khan, r. Burns, and g. Ateniese, “mr-pdp: multiple-replica provable data possession,” in proc. 28th int. Conf. Distributed comput. Syst. (icdcs), beijing, china, jun. 2008, pp. 411–420. Available from: https://ieeexplore.ieee.org/abstract/document/4595910

[4] H. Shacham and b. Waters, “compact proofs of retrievability,” in advances in cryptology – asiacrypt 2008, j. Pieprzyk, ed., vol. 5350, lecture notes in computer science. Berlin, germany: springer, 2008, pp. 90–107. Available from: https://link.springer.com/article/10.1007/s00145-012-9129-2

[5] C. Erway, a. Küpçü, c. Papamanthou, and r. Tamassia, “dynamic provable data possession,” in proc. 16th acm conf. Comput. Commun. Secur. (ccs), new york, ny, usa, 2009, pp. 213–222. Available from: https://doi.org/10.1145/2699909

[6] K. D. Bowers, a. Juels, and a. Oprea, “hail: a high-availability and integrity layer for cloud storage,” in proc. 16th acm conf. Comput. Commun. Secur. (ccs), new york, ny, usa, 2009, pp. 187–198. Available from: https://doi.org/10.1145/1653662.1653686

[7] Q. Wang, c. Wang, j. Li, k. Ren, and w. Lou, “enabling public verifiability and data dynamics for storage security in cloud computing,” in proc. 14th eur. Conf. Res. Comput. Secur. (esorics), berlin, germany, 2009, pp. 355–370. Available from: https://link.springer.com/chapter/10.1007/978-3-642-04444-1_22

[8] C. Wang, q. Wang, k. Ren, and w. Lou, “privacy-preserving public auditing for data storage security in cloud computing,” in proc. 29th annu. Ieee conf. Comput. Commun. (infocom), san diego, ca, usa, 2010, pp. 525–533. Available from: https://ieeexplore.ieee.org/abstract/document/5462173

[9] B. Chen, r. Curtmola, g. Ateniese, and r. Burns, “remote data checking for network coding-based distributed storage systems,” in proc. 2010 acm cloud comput. Secur. Workshop (ccsw), new york, ny, usa, 2010, pp. 31–42. Available from: https://doi.org/10.1145/1866835.1866842

[10] G. Ateniese, r. Burns, r. Curtmola, j. Herring, o. Khan, l. Kissner, z. Peterson, and d. Song, “remote data checking using provable data possession,” acm trans. Inf. Syst. Secur., vol. 14, no. 1, pp. 12:1–12:34, jun. 2011. Available from: https://doi.org/10.1145/1952982.1952994

[11] N.-y. Lee and y.-k. Chang, “hybrid provable data possession at untrusted stores in cloud computing,” in proc. 17th ieee int. Conf. Parallel distrib. Syst. (icpads), Washington, DC, USA, 2011, pp. 638–645. Available from: https://ieeexplore.ieee.org/abstract/document/6121335

[12] F. Liu, d. Gu, and h. Lu, “an improved dynamic provable data possession model,” in proc. Ieee int. Conf. Cloud comput. Intell. Syst. (ccis), beijing, china, sept. 2011, pp. 290–295. Available from: https://ieeexplore.ieee.org/abstract/document/6045077

[13] Z. N. J. Peterson, m. Gondree, and r. Beverly, “a position paper on data sovereignty: the importance of geolocating data in the cloud,” in proc. 3rd usenix conf. Hot topics cloud comput. (hotcloud), berkeley, ca, usa, 2011, p. 9. Available from: https://tinyurl.com/4h9zbf46

[14] Q. Zheng and s. Xu, “fair and dynamic proofs of retrievability,” in proc. 1st acm conf. Data appl. Secur. Privacy (codaspy), new york, ny, usa, 2011, pp. 237–248. Available from: https://doi.org/10.1145/1943513.1943546

[15] B. Chen and r. Curtmola, “robust dynamic provable data possession,” in proc. 32nd int. Conf. Distributed comput. Syst. Workshops (icdcsw), macau, china, jun. 2012, pp. 515–525. Available from: https://ieeexplore.ieee.org/abstract/document/6258200

[16] A. F. Barsoum and m. A. Hasan, “integrity verification of multiple data copies over untrusted cloud servers,” in proc. 12th ieee/acm int. Symp. Cluster, cloud grid comput.

(ccgrid), washington, dc, usa, 2012, pp. 829–834. Available from: https://ieeexplore.ieee.org/abstract/document/6217519

[17] M. Krzywiecki and m. Kuty, "proof of possession for cloud storage via lagrangian interpolation techniques," in proc. 6th int. Conf. Network syst. Secur. (nss), berlin, germany, 2012, pp. 305–319. Available from: https://link.springer.com/chapter/10.1007/978-3-642-34601-9_23

[18] A. Mohan and r. Katti, "provable data possession using sigma-protocols," in proc. 11th ieee int. Conf. Trust, secur. Privacy comput. Commun. (trustcom), liverpool, uk, jun. 2012, pp. 565–572. Available from: https://ieeexplore.ieee.org/abstract/document/6296021

[19] S. Sundareswaran, a. Squicciarini, and d. Lin, "ensuring distributed accountability for data sharing in the cloud," ieee trans. Dependable secure comput., vol. 9, no. 4, pp. 556–568, jul. 2012. Available from: https://ieeexplore.ieee.org/abstract/document/6165313

[20] B. Wang, b. Li, and h. Li, "knox: privacy-preserving auditing for shared data with large groups in the cloud," in proc. 10th int. Conf. Appl. Cryptogr. Netw. Secur. (acns), berlin, germany, 2012, pp. 507–525. Available from: https://link.springer.com/chapter/10.1007/978-3-642-31284-7_30

[21] S. Worku, z. Ting, and q. Zhi-guang, "survey on cloud data integrity proof techniques," in proc. 7th asia joint conf. Inf. Secur. (asia jcis), tokyo, japan, aug. 2012, pp. 85–91. Available from: https://ieeexplore.ieee.org/abstract/document/6298140

[22] Q. Zheng and s. Xu, "secure and efficient proof of storage with deduplication," in proc. 2nd acm conf. Data appl. Secur. Privacy (codaspy), new york, ny, usa, 2012, pp. 1–12. Available from: https://doi.org/10.1145/2133601.2133603