A Scalable Data Integrity Mechanism Based on Provable Data Possession and JARs

Faheem Zafar[1], Abid Khan[*1], Mansoor Ahmed[1], Majid Iqbal Khan[1], Farhana Jabeen[1], Zara Hamid[1], Naveed Ahmed[1], and Faisal Bashir[2]

[1] COMSATS Institute of Information technology, Computer Science Department, Park Road Chak Shahzad, Islamabad, Pakistan
[e-mail: faheemiiui@gmail.com, abidkhan@comsats.edu.pk , mansoor@comsats.edu.pk, majid_iqbal@comsats.edu.pk, farhanakhan@comsats.edu.pk, zarahamid@comsats.edu.pk naveedahmad@comsats.edu.pk]

[2] Bahria University, Computer Science Department, Islamabad, Pakistan
[E-mail : faisalbashir@bahria.edu.pk]

*Corresponding author: Abid Khan

Received July 20, 2014; revised August 05, 2015; accepted May 04, 2016; published June 30, 2016

Abstract

Cloud storage as a service provides high scalability and availability as per need of user, without large investment on infrastructure. However, data security risks, such as confidentiality, privacy, and integrity of the outsourced data are associated with the cloud-computing model. Over the year's techniques such as, remote data checking (RDC), data integrity protection (DIP), provable data possession (PDP), proof of storage (POS), and proof of retrievability (POR) have been devised to frequently and securely check the integrity of outsourced data. In this paper, we improve the efficiency of PDP scheme, in terms of computation, storage, and communication cost for large data archives. By utilizing the capabilities of JAR and ZIP technology, the cost of searching the metadata in proof generation process is reduced from $O(n)$ to $O(1)$. Moreover, due to direct access to metadata, disk I/O cost is reduced and resulting in 50 to 60 time faster proof generation for large datasets. Furthermore, our proposed scheme achieved 50% reduction in storage size of data and respective metadata that result in providing storage and communication efficiency.

Keywords: Data Integrity, Provable Data Possession, Storage as a Service (SaaS), Storage Security, Scalability

http://dx.doi.org/10.3837/tiis.2016.06.022 **ISSN : 1976-7277**

1. Introduction

Cloud Computing provides multiple services with elasticity, agility, multi-tenancy and high availability as per need of user. The deployment models used by cloud computing are Private Cloud (internal datacenters of business or organizations which are not available for general public use), Public Cloud (private cloud made accessible for general public use on pay-as-you-go model), Hybrid/Multi Cloud (cloud computing environment in which an organization provides and manages some resources in-house, and the other services are provided externally) . Cloud computing services are organized as IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). Small organizations with large secondary storage requirements are often unable to maintain data centers, because of the cost associated with building and managing such infrastructure. Managing large tape library for archival data is a cumbersome job. Storage as a Service (SaaS) appeals to such users because of its flexible model. Multiple storage services are available today, such as Dropbox, Skydrive, Amazon S3, Googledrive, Box, and Sugar Sync. When data storage is outsourced, user loses control of data. Although cloud service provider (CSP) is bounded by service level agreement (SLA) to ensure data security, but users cannot solely rely on such agreements. Besides the convenience provided by the model, data security issues, such as confidentiality, privacy, and data integrity are associated with SaaS. Therefore, data security assurance mechanism is utmost requirement for SaaS adoption, which will enable users to frequently check integrity of the data in secure and efficient manner.

Data integrity mechanism should be cost effective and robust so that any corruption or deletion can be timely identified by data owner. Security of data integrity mechanism itself is also very important as it must resist against attacks like data leakage attack or tag forgery attack. In data leakage attack, an adversary using techniques like wiretapping enough of communications extracts stored data through verification protocol. In tag forgery attack a client is deceived by a dishonest service provider. Techniques like provable data possession (PDP) [1,2,3,5,9-14,16,19-24,26], remote data checking (RDC)[12,20], data integrity checking or data integrity protection (DIP)[7], proof of storage (POS) [18], and proof of retrievability (POR) [4,8] exists to serve this purpose.

Data integrity schemes have a common methodology of generating some metadata using original data, which later on is used for integrity verification. Focus of this research is how to make the existing PDP [1] scheme more efficient in terms of computation, storage, and communication cost and makes it applicable to large archives. Data integrity checking schemes can be classified into two main categories: (1) Provable Data Possession (PDP), (2) Proof of Retrievability (POR). PDP schemes are probabilistic because these use sampling random blocks instead of reading whole file for verification. In PDP, original data is preprocessed to generate some metadata, which is outsourced with original data. This metadata is later on used to verify the integrity of user's data. These schemes can only identify the corruption in data but there is no support for recovery of corrupted data. Proof of Retrievability (POR) scheme is very much similar to PDP but it also provides the data recovery. POR scheme uses the redundant encoding of data and thus provides the recovery in case of failure. In other words, PDP scheme can be transformed to POR by using error correcting or erasure codes. Auditing protocol of POR provides the guarantee that CSP is holding all data of client and that is still retrievable. However PDP only ensures that server is holding most of the data of a particular client, because of the probabilistic nature of the

algorithm used in verification phase. When data is outsourced, then complexity of proof generation process is also delegated to the cloud. Only preprocessing computation complexity is kept at the data owner side. However, preprocessing overhead is only one time cost at the time when the data is outsourced to CSP.

In this paper, classical PDP scheme proposed in [1] by Ateniese et al is chosen, because our focus is on the integrity of archival data, which is considered as append only or static data being accessed very rarely. Strength of this scheme is, by inspecting just 4.6% of total blocks of file it can provide 99% possession guarantee. This scheme is data format independent and is applicable for all types of data. Client side storage is O(1), challenge and response size is approximately 1KB. Thus, it is efficient in terms of client side storage and communication, however performance of proof generation process in this scheme degrades with increase in size of files.

1.1 Provable Data Possession (PDP)

Ateniese et al.[1] proposed first formal probabilistic PDP model using sampling of random blocks for data integrity verification without retrieving the data from server. This technique greatly reduces I/O cost by accessing file partially for proof generation instead of complete file access. PDP system model is based on two entities

1. Data Owner/Client
2. Storage Server/Cloud Storage Provider (CSP)

Before outsourcing the data over the cloud, data owner preprocess the file and generates the homomorphic verifiable tags. Homomorphic verifiable tags (HVT) are used for data integrity verification by PDP [1] scheme. These tags have homomorphic property by which multiple tags are computed to generate a single value, which serves as a proof of possession.

In original PDP construction, HVT consists of the following pair of values: $T_{i,m}$ and W_i. Wi is a random value which is obtained from an index i and $T_{i,m}$ is stored at the server. The index i is never reused for computing tags, which ensures that every tag uses a different index i. To ensure that W_i is different and unpredictable each time a tag is computed, W_i is generated by concatenating the index i to a secret value. Compared to the actual file blocks HVTs and their corresponding proofs have a much smaller fixed constant size.

Therefore with the help of these HVT data owner is able to perform blockless verification. To verify the integrity of data, client generates the random challenge by choosing random blocks of file. For these randomly chosen blocks indexes, server accesses the block's data and respective HVTs and computes the proof of possession. This proof is provided to data owner for verification and thus client is able to verify the integrity of his data without retrieving the file from server. After verification, client may permanently delete the data from client machine. Therefore integrity verification of data over storage server is necessary. Protocol of provable data possession (PDP) [1] is shown in Fig.1.

PDP scheme's working is independent of data format. It is applicable to all types of data either in plaintext or binary form. PDP scheme does not propose encryption of data itself but it works even if data is encrypted. Another important factor for practical application of scheme is capability of unbounded queries. It means that there should be no limit on how many times data owner can ask the server to provide proof of possession. PDP scheme is probabilistic as it works on sampled blocks. It does not provide deterministic guarantee of possession because

for that it requires accessing all blocks of file. Even with sampling, possession guarantee is with very high probability. Any corruption or unauthorized deletion of blocks can be detected by querying fixed number of blocks irrespective of how many total blocks of file are. PDP claims that if file has 10,000 blocks and server has deleted only 1% of these blocks then by querying only 460 randomly chosen blocks in proof generation, misbehavior of server can be detected with high probability of 99%. This probabilistic approach of PDP provides performance along with strong guarantee.

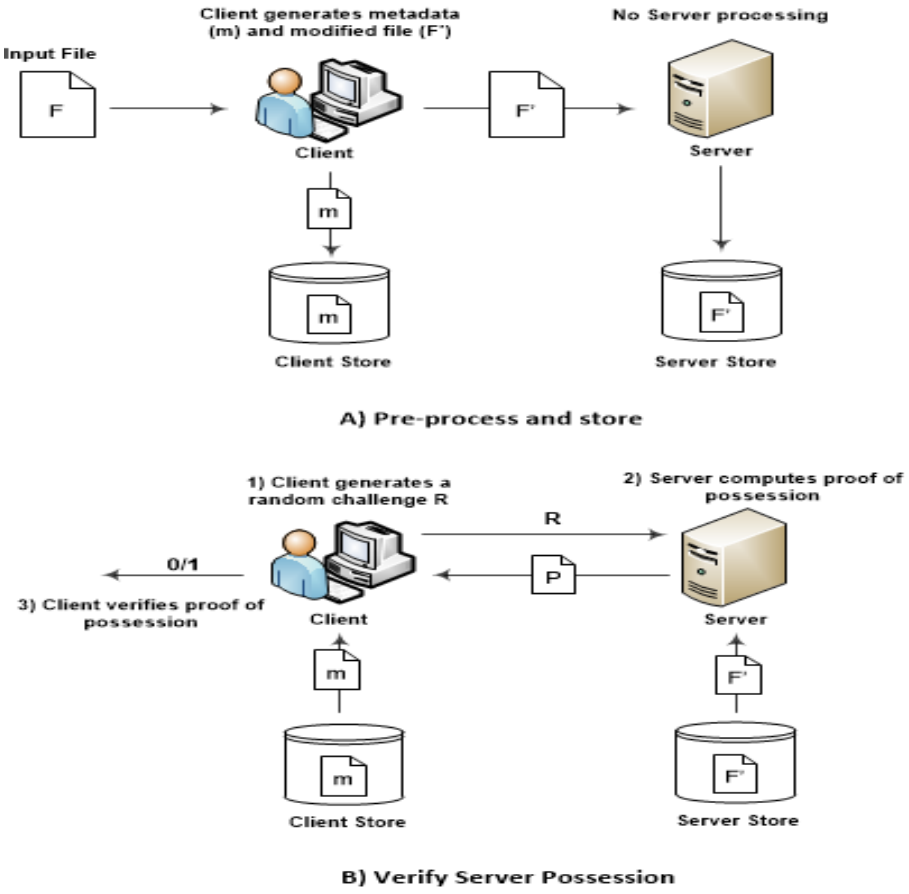

Fig. 1. Protocol for Provable Data Possession (PDP) [1]

Organization of the paper: In Section 2, we provide problem statement and desirable characteristics of data integrity mechanism; also a description of performance degrading parameters is given. Review of existing work is provided in the Section 3. Section 4 describes our approach of data integrity based on PDP and JARs, providing the details of preprocessing and proof generation algorithms. In section 5 we describe experiments with proposed approach and give a comparison with existing technique.

2. Problem Statement

To build the trust of users in cloud computing technology we need a robust, secure, and

reliable data integrity mechanism for outsourced data. The mechanism should not only be secure but should also be efficient and cost effective in terms of computation, communication, and storage. Existing PDP schemes met most of the desirable characteristics of data integrity schemes, but are unable to meet the scalability requirement. When existing scheme is evaluated for large datasets, we observe that there is a substantial degradation in performance and increase in the cost of the scheme. It is quite challenging to come up with a scheme that can handle all the performance-degrading factors in a way such that performance is not affected. However, well handling of the factors can make existing scheme more efficient. Based on the analysis, we proposed an approach that minimizes the impact of the aforementioned factors and makes the existing scheme more effective in terms of performance and cost.

2.1 Desirable characteristics of Data Integrity Schemes

A data integrity mechanism must provide the following properties [1]:

1. Data owner must be able to verify the integrity without retrieving the whole file from server.
2. Verification process must be secure i.e. not deceivable by service provider.
3. Verification must be computational efficient in terms of resources and time.
4. Metadata size must be small as compared to data.
5. Verification process should guarantee strong integrity checking without accessing all blocks of file.
6. No restriction for data owner to verify the integrity of data multiple times.
7. Challenge and response phase of scheme should be efficient in communication. Size of challenge request and proof of possession in response must not be very large in size.
8. Scheme should be scalable i.e. it should be applicable for large datasets having size in GBs.

So a mechanism has to be designed, which should have all these properties.

2.2 Performance Degrading Parameters of PDP

Performance of PDP scheme is affected by following parameters

- **Variable length of metadata:** Due to variable length, metadata for particular block index cannot be directly accessed. The searching of metadata tag is similar to searching in linked list. To reach a particular index's metadata tag, metadata file has to be traversed sequentially by skipping undesired tags. The traversal process increases disk I/O as number of block tags increases in metadata file, restuling in increase in proof generation time.
- **Unordered block indexes in challenge:** If indexes of blocks are in ascending order in challenge generated by client, then verification time can be improved. The searching for each metadata tag in challenge will not require the traversing of metadata file from start and the server can continue to look for next index in challenge from current position in metadata file.

- **Size of block:** Smaller the size of block more will be the number of blocks in file. The block size will impact preprocessing time in generating metadata tags and also proof generation time due to variable length of metadata.
- **Number of blocks in challenge:** More the number of blocks in challenge, more time will be required to generate the proof.

The focus of the research work is to make the existing data integrity verification scheme more efficient in terms of computation, storage, and communication for large data archives and minimizes the impact of performance degrading parameters to make it scalable. JAR allowed us to preprocess the data in such a way that searching cost of metadata tags is reduced from $O(n)$ to $O(1)$. As a result, disk I/O is decreased and performance is increased.

3. Related Work

All existing schemes of data integrity checking rely on some metadata, which is generated using original data before it is outsourced. Later on, using this metadata, proofs for integrity verification are generated by cloud service provider (CSP) and verified by data owner. Research work on data integrity checking mechanism started from static data or append only data, and then extended to dynamic data (supporting CRUD operations at block level) and concept of private (single user) and public (multiple user) verifiability were introduced with or without third party auditor (TPA). Privacy issues were identified due to introduction of TPA and multiuser, so techniques of data integrity checking mechanism with privacy preservation were introduced. Applications of these techniques were extended to multiple copies of data [3], constraint-based data geolocation (CBDG) [22], multi-cloud (cloud of clouds) [19], data deduplication and proof of ownership [18]. Techniques such as Homomorphic tagging [1], bilinear pairing [25], algebraic signatures [20], zero knowledge proofs [26], fountain codes, erasure codes, RS codes based on Cauchy matrices and other techniques as metadata for integrity verification were developed.

In 2007, Ateniese et al. [1] proposed first formal PDP model. This is a probabilistic data integrity verification technique for cloud users, who can verify the integrity of the outsourced data without retrieving the data. However, the model lacked support for dynamic operations. Later on, Ateniese et al. [2] extended this model with support for dynamic data operations like update and delete but support for insertion was still missing. Erway et al. [5] proposed a fully dynamic PDP scheme with data insertion capability using rank-based authenticated skip lists. Dynamic PDP schemes, which depend on block indexes in dynamic scenarios, proved to be insecure against replay attacks and are made secure by use of authenticated data structures e.g. authenticated skiplist. Shacham and Waters [4] proposed two POR schemes. The former is based on BLS signatures and provides public verifiability and scheme allows short query with short response. The later scheme used pseudorandom functions (PRFs) for handling longer query with short response and provides private verifiability only and didn't support dynamic data operations. Moreover, in the proposed scheme the verification protocol was unable to prevent data leakage. Bowers et al. [7], in High-Availability and Integrity Layer (HAIL) scheme, suggested integrity-protected error-correcting code (IP-ECC) for POR. IP-ECC construction is based on PRFs, ECCs, and universal hash functions (UHFs). HAIL served as corruption-resilient MAC for data, providing strong security against active and mobile adversaries. HAIL also provides high availability using both within-server redundancy and

cross-server redundancy. Curtmola et al. [3] extended PDP for provably secure multiple copies in multiple-replica PDP (MR-PDP) scheme, consisting of two approaches. First is (Single-Replica PDP) addresses single file replica, which is insecure against collusion attack. The second scheme is ENC-PDP, which is secure against collusion attack.

Wang et al. [9] proposed improved POR based on Merkle Hash Tree (for BLS based block tag authentication). Main characteristics included public verifiability by third party auditor (TPA) and dynamic data storage operations. However, cryptanalysis performed in [26] proved that scheme does not guarantee knowledge soundness, which means that a dishonest server can deceive the client by passing the audit without holding the data. A privacy preserving dynamic PDP is proposed in [10], using homomorphic authenticator with random masking supporting public verifiability. Using bilinear aggregate signature, the complexity of O(n) for verification of data by TPA was achieved. Zhu et al. [19] examined characteristic of high security, high performance and transparent verification for PDP model in context of multicloud, and proposed a framework using hash index hierarchy (HIH), homomorphic verifiable response, and multiprover zero-knowledge proof system, which can resists against data leakage and tag forgery attacks. In [21], Ertem et al. proposed FlexList based on authenticated skiplist for PDP, which achieved variable block size updates instead of fixed length blocks. In [22], Zachary et al. proposed constraint-based data geolocation using latency-based techniques applying PDP for binding data to location. However the proposed scheme is not suitable, if a private network of significantly better quality than the Internet connects remote sites.

3. Scalable data integrity mechanism based on PDP and JARs

The proposed scheme uses the same homormorphic verifiable tags(HVT) as proposed by PDP [1] as metadata. So we make the same security assumptions for HVTs as claimed in original PDP [1]. However our proposed approach place the block and relevant metadata together in a single zip file. Saving the individual metadata tags in separate files doesnot have security or integrity issue of tags themselves. Because data integrity schemes are temper evident not temper resistant. If any tag file is deleted/modified it will be identified in verification process. Once all blocks are processed and respective zip files are generated these zipped files are embedded in a single JAR. We make sub-files of one big file such that individual blocks and their respective metadata are embedded in single sub-file (zip file) and these individual zips will be embedded in single JAR file. For individual zip files, index of block is used as file name e.g. for block at index 1, zip file name is "z1.zip". Later on in verification process, we can directly access each block for particular index and thus we will have direct access to both block data and respective metadata without any computation. Thus reducing the disk I/O cost from O(n) to O(1) for metadata access in verification step. I/O cost reduction from O(n) to O(1) can be better understood with example of array and linked list data structure. In array we can directly access any index and in case of linked list we have to traverse nodes to reach desired node. So if indexes, to be accessed from linked list are not in sorted order then for each index, we have to traverse linked list from start but if indexes to be accessed are in sorted order then we do not need to traverse the linked list repeatedly from start. But for array access order has no impact as all indexes are directly accessible. Same is the case in our JAR approach. Each block and respective metadata is in zip file having its index as name. And as JAR has already computed hashes for all these zips embedded in it. On access to any of these zip files by their name, in any order, during verification process have same constant cost treating as

O(1).

Programmable JAR provides the object-centered approach by encapsulating the data and control logic together [17]. JAR allowed us to reduce the searching cost of metadata tags from O(n) to O(1). Preprocessing of data and making of JAR is shown in Figure 3. One of the motivations of our work is that we want to leverage the JAR programmable capabilities to both create a dynamic and traveling object, a capability that is missing in ZIP technology [17]. JAR signing capabilities could be utilized for providing secure computation assurance, which is not our concern here. Our focus is only on improving performance of existing PDP scheme. Due to this, disk I/O is decreased and performance is increased. JAR also includes the functionality that original data file can be recovered anytime. JAR based approch provides the priviledge for computation assurance [27] which is another future aspect of research in data integrity schemes. First assumption of computation assurance [27] is that CSP is untrusted and computation reliability is serious concern but in this paper our focus is only on performance. JAR signing capabilities could be utilized for providing secure computation assurance. We are working on utilizing JAR based approach for computation assurance and geolocation assurance [28,29] of out sourced data along with data integrity verification. But both computation assurance [27] and geolocation assurance [28,29] are out of scope for this paper. This paper will server as base for our future work in data integrity schemes with geolocation location and computation assurance.

Proposed scheme is divided into two phases. Phase 1 includes the preprocessing of input file, jar file generation, and transferring it to cloud. In phase 2, data owner generates a challenge; containing fix number of randomly chosen file block indexes from all blocks of file. Data owner sends the challenge to the server to provide proof of possession against indexes in challenge. Server generates the proof of possession for block indexes in challenge and sends it back to data owner for verification. Data owner then verifies the proof to ensure that data is intact. Figure 2, shows the flow of our proposed PDP scheme derived from PDP scheme of Ateniese et al in [1].

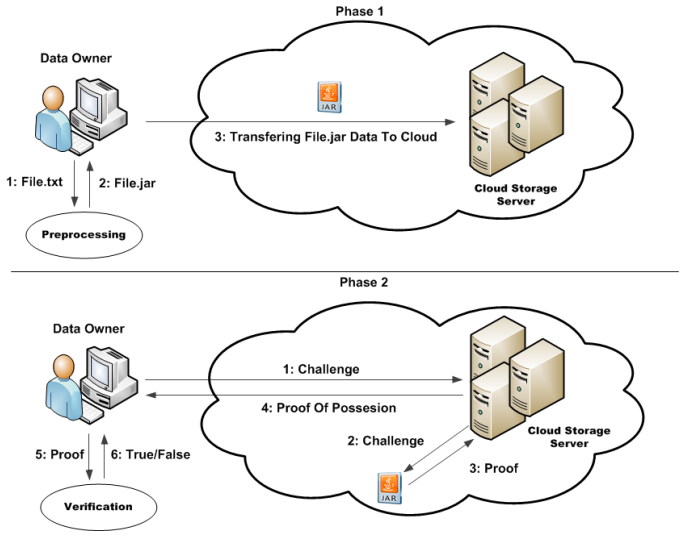

Fig. 2. Proposed PDP Scheme Flow

Fig. 3, describes the preprocessing algorithm working on data and making of JAR

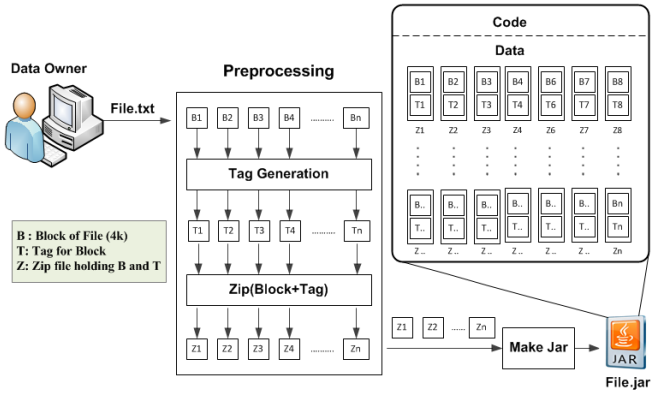

Fig. 3. Proposed PDP Scheme Preprocessing

Fig. 4, gives the internal details of working of proposed approach

Step 1: Data owner preprocess the input data to produce JAR, which is to be placed over the cloud. In preprocessing step, data file is divided into fixed length blocks and for each block metadata tag is generated. Both block and tag are zipped together making a single zip file. Index of the block is used as the name of the zip and then zip file is added to JAR.

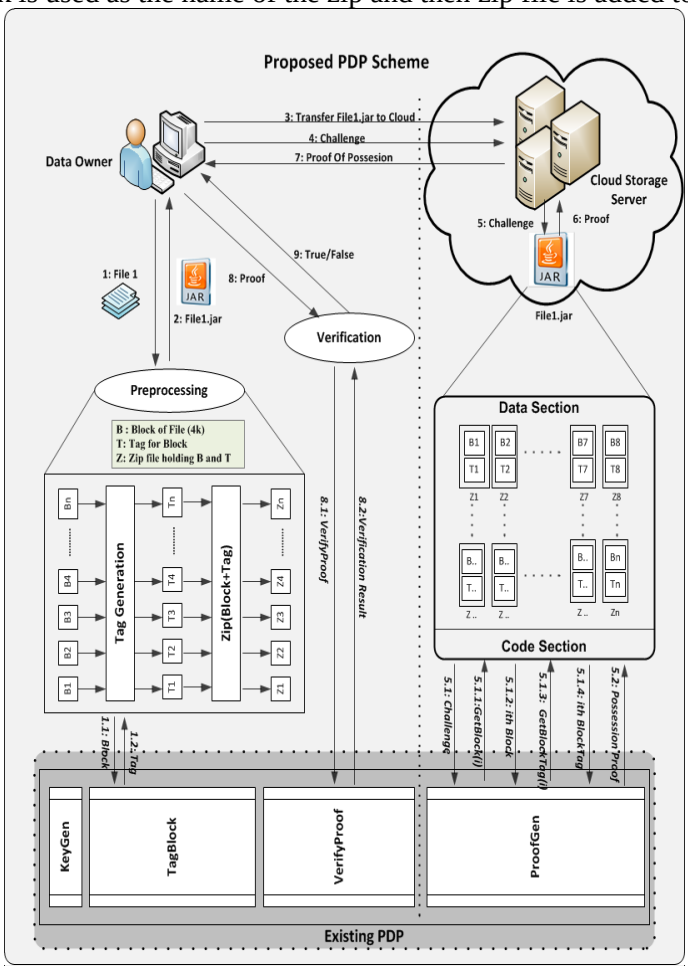

Fig. 4. Proposed Scalable PDP Scheme

Step 2: Once all data is preprocessed and added to JAR, executable code is added to it, which allows accessing the data and metadata during proof generation over the storage server.
Step 3: Finalized JAR containing both data and code is transferred to storage server over the cloud.
Step 4: Once data is placed over the cloud, data owner sends a challenge containing randomly chosen sampled block indexes to storage server to verify the integrity of data.
Step 5: Storage server then passes the challenged blocks information to JAR by invoking the proof generation code. JAR file extracts the block contents and respective metadata tags for the indexes in challenge and generates the proof. Since we have used the same construction of homomorphic tags, proof generation, and verification proposed by PDP scheme of Ateniese in [1], it is presented as Existing PDP in figure.
Step 6: After utilizing all the block indexes in challenge, final proof is returned by JAR to storage server.
Step 7: Proof generated by JAR on storage server is then returned to data owner for verification.
Step 8: Data owner verifies the proof of possession provided by storage server by using the same verification method of Ateniese PDP scheme. If any of block is modified, deleted or additional block is added then verification will fail otherwise it will succeed.
Step 9: Result of verification process is returned to data owner in notified to data owner.

Our proposed PDP scheme is based on following algorithms

- **GenKey(1k) → (pk, sk)** is a key generation algorithm executed on client. Working of PDP depends on these keys. Its argument is a security parameter k, and it returns a pair of matching public and secret keys (pk, sk).

- **PreprocessFile(file) → jar** treats the input file as a collection of blocks and generates the homomorphic tags for all blocks. Each block and respective tag is zipped in single file and added to JAR.

Algorithm 1 Preprocess Input File And Make Jar

```
1:  procedure PREPROCESSFILE(file)          ▷ Input file
2:  BLOCK_SIZE = 4096                       ▷ block size 4k
3:  offset = 0                              ▷ bytes offset
4:  i = 0                                   ▷ block index
5:  jar = CREATEJAR()
6:  B_i = READFILE(file,offset,BLOCK_SIZE)
7:  while B_i ≠ EOF do      ▷ till end of file
8:      T_i = EXISTINGPDP.TAGBLOCK(B_i,i )
9:      Z_i = MAKEZIP(B_i,T_i )
10:     jar.ADDTOJAR(Z_i)
11:     offset = offset + BLOCK_SIZE
12:     i = i +1
13:     B_i = READFILE(file,offset,BLOCK_SIZE)
14: end while
15: return jar
16: end procedure
```

Algorithm 1 describes the preprocessing steps of proposed PDP scheme shown in **Fig. 3**. It takes an input file and treats it as collection of 4k bytes blocks. Each block (B_i) is identified by its index (i). Tag (T_i)is generated using original PDP of Ateniese et al, against each block using its index and data bytes. Both data bytes and tag are zipped together in a zip file (Z_i). Index of the block is used as the name of the zip file e.g. "1.zip" which helps in proof generation time to find particular index block and tag as absolute path of the zip file is known in JAR. This process is repeated for all blocks of input file. A JAR file is created having code and data section. Code section contains the necessary code that provides the access to data and metadata in proof generation process while data section holds the zip files produced against all file blocks. This approach provides direct access to both data bytes and tag for particular index.

- **GenerateProof(jar,C) → V** is executed on server on the request of client. The arguments are jar file as an input holding ordered collection of blocks and tags, a challenge C. It generates the proof of possession V for the challenged blocks by accessing blocks and tags inside the jar.

Algorithm 2 Generate Proof Of Possession

```
1:  procedure GENERATEPROOF(jar,C)          ▷ C, challenged blocks
2:  i = -1                                  ▷ challenged block index
3:  p = EXISTINGPDP.CREATEPROOF()
4:  for index = 0 to C.size() – 1 do
5:      i = C[index]
6:      B_i = jar.GETBLOCK(i)
7:      T_i = jar.GETBLOCKTAG(i)
8:      p = EXISTINGPDP.UPDATEPROOF(B_i,i )
9:  end for
10: return proof
11: end procedure
```

Algorithm 2 describes the steps, for generating the proof of possession on cloud storage server with the help of JAR. A challenge (C) is an unordered collection of sampled block indexes and jar is expected file for which storage server has to provide proof of possession. Server gets data block (B_i) and tag (T_i) against challenged block index (i) from JAR and uses it for proof generation (using original PDP [1]). This process is repeated for all blocks indexes specified in challenge and at the end final proof of possession is returned. Our proposed scheme does not modify internal working of original PDP scheme proposed in [1]. The only change is that it relies on JAR to provide data block and tag for particular challenged index instead of reading itself. Each block and respective tag are zipped in single file and the name is index of block, and is accessible without any extra computation. This approach reduces disk access largely and increases performance immensely.

- **VerifyProof(V,C) → {"true", "false"}** is executed by the data owner to verify the proof provided by server. The arguments are challenge C and a proof of possession V. The verification of proof is successful means that the server provided a valid proof for

challenged blocks and returns true and if proof is invalid, it will return false indicating the misbehavior of the server.

One unique characteristic of proposed approach is that it does not change the internal mechanism of metadata generation and data integrity verification. It also does not introduce any confidentiality or privacy risks and achieves about 50% reduced size of data and respective metadata collectively. The results show an increase of 40 to 60 times fast execution in computation of proofs for large archives.

4. Experimental Evaluation

To evaluate the proposed scheme, we conducted multiple tests using a commodity machine having Ubuntu 12.04 (OS) with 3GB RAM, 80GB HDD and 2 Processors (2 cores each) with Java 1.7 installed. We analyzed the schemes for storage and communication cost, verification and preprocessing time. The results have been presented in the form of graphs. In graphs, OLD_PDP represents the PDP scheme of Ateniese et al [1], NEW_PDP represents proposed scheme, whereas "Output" refers to collection data and metadata which is to be outsourced.

4.1 Storage and Communication Cost Analysis

Storage and communication cost matters in scheme evaluation, as one has to pay for both storage and bandwidth utilization for transferring data over cloud. The size of data along with metadata has impact on the transfer time. Therefore, we have evaluated both schemes for output size, which includes the data, and respective metadata tags file. We have performed tests on multiple files by increasing size of file gradually. We performed tests on large files using a data block size of 4KB as worst case and keeping metadata tags size as variable length.

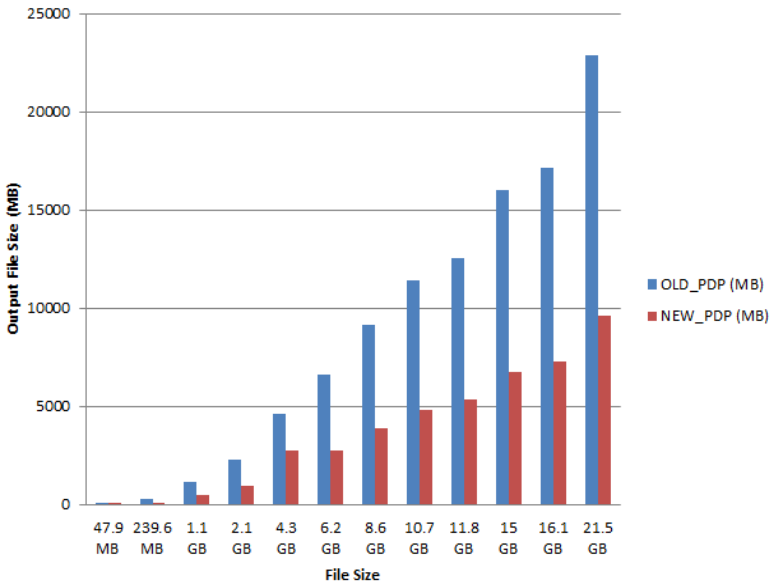

Fig. 5. Scheme's Output Size Comparison

Comparison of the output size for both schemes clearly shows that proposed scheme achieved more than 50% reduction in size, resulting in cost effectiveness in terms of storage and communication. The communication time and bandwidth utilization for data transfer is less in NEW_PDP as compared to OLD_PDP. The same construction proposed by Ateniese et al [1], for metadata tags generation is used, so tags in both schemes are exactly same for same input data block. Therefore, reduced size is not because of metadata size. Reason for the reduced size is use of ZIP technology as we are placing the block contents and respective metadata tags for each index in zip in NEW_PDP. In case of OLD_PDP scheme, it is maintaining the data and metadata tags in normal uncompressed files so output size of OLD_PDP is greater than output size of NEW_PDP. This increase in size impacts the communication time and cost as well as storage cost over the cloud. The output size can be improved for both schemes by increasing the data block size from 4KB.

It is obvious that using smaller block size results in larger number of blocks for file. In our experimentation, we kept block size to be 4KB as the smallest block size of data. Increase number of blocks, results in the number of metadata tags to be very large. Therefore, the size of metadata tags file will be large in this case and result in increasing the resultant output file to be outsourced. Output size can be improved by using large block size (e.g. 8KB or 16KB), as it will decrease the number of total blocks of file. The impact of block size parameter will affect both schemes. Our proposed approach can benefit from a larger block size.

4.2 Verification Process Time Analysis

Data owner generates a challenge containing fixed number of block's indexes, randomly chosen from all blocks of file. Therefore, the cloud storage server has to provide proof of possession of data against block indexes in challenge. Also the server needs to access the data block contents and respective metadata tags for challenged indexes. The time taken by the server to compute the proof of possession for both schemes is compared for different file sizes as shown in fig.6. We have used data block size of 4KB, keeping metadata tags size as variable length, number of blocks in challenge as 460, and block indexes in challenge as unsorted order.

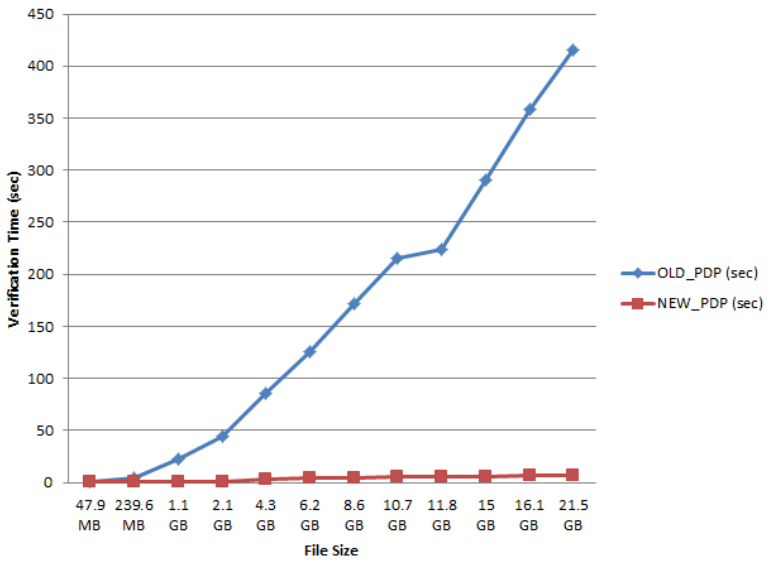

Fig. 6. Schemes Verification Time Comparison

Since 4KB data block size is very small, total number of blocks of file increases with increase in file size and proof generation time for 460 randomly chosen blocks in challenge also increases. Comparison of verification time in fig.6 shows that NEW_PDP performance is better than existing scheme. As data file size grows, the OLD_PDP scheme performance degrades, whereas proposed scheme is 50 to 60 times faster in proof generation. In existing scheme, more disk I/O is performed to search the metadata tags for each index in challenge. Variable length of metadata tags, size of data block, and unordered indexes of blocks in challenge limit performance of OLD_PDP scheme.

In OLD_PDP scheme, for each index in challenge, server can directly access the data block contents as data blocks are of fixed length, but tags are of variable length. Therefore, to read tag of particular index in challenge, server has to traverse metadata file from start and keep on skipping the tags until reaches to the desired index tag. This behavior is similar to searching in linked list as you have to traverse the linked list nodes to reach the node at particular index. This process is repeated for each index in challenge and it increases the proof generation time due to increased I/O cost of disk accesses. But in NEW_PDP, variable length of metadata does not impact the performance as metadata tags for all indexes are directly accessible by the absolute paths in JAR. This behavior is similar to direct access in arrays and due to reduced disk I/O, performance is better in NEW_PDP scheme.

Performance of OLD_PDP scheme can be improved, if indexes in challenge are in sorted order. Due to unsorted order, impact of variable length of tags get drastic. Because the same process of searching the tags will be repeated from start of metadata file for every index in challenge. If indexes are in sorted order, the server does not need to start search from beginning of file and may continue searching from current position for next index in challenge. If more blocks indexes in challenge are from end of the file, means higher indexes, the verification time of OLD_PDP also increases in this case. Similarly, if block indexes are from start of file, it means lower number indexes, the verification time will improve as less I/O is performed to locate the tags in metadata file. In NEW_PDP, all tags are directly accessible so order of indexes in challenge has no impact on it.

To see the impact of number of blocks in challenge on verification time, we performed tests by keeping the file size fixed and increasing the number of blocks in challenge gradually. To ensure the scalability of our proposed scheme we performed experiments on two files. First test is performed using file size 500 MB

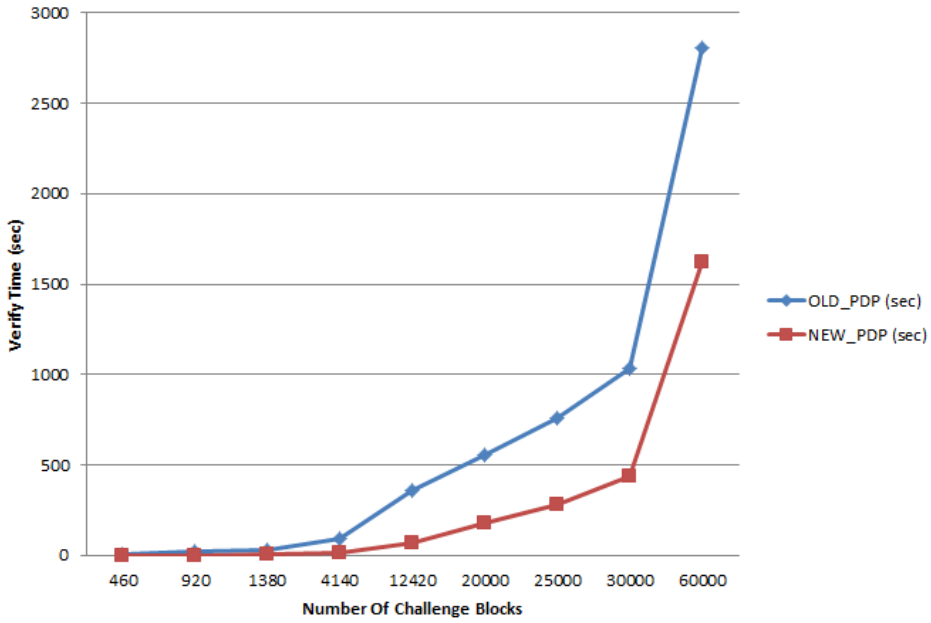

Fig. 7. Challenge Blocks Imapct on Verification Time for 500 MB File

By analyzing **Fig. 7**, it has been observed that proposed scheme is better in proof generation but trend of verification time is the same as that of existing scheme.
It is observed that the difference between the time of verification process of proposed scheme and original scheme is quite large for huge archival files. We executed same test on 4 times larger file to see how both parameters (no. Of blocks in challenge, file size) impact the verification time. For second test we used file size 2.1 GB

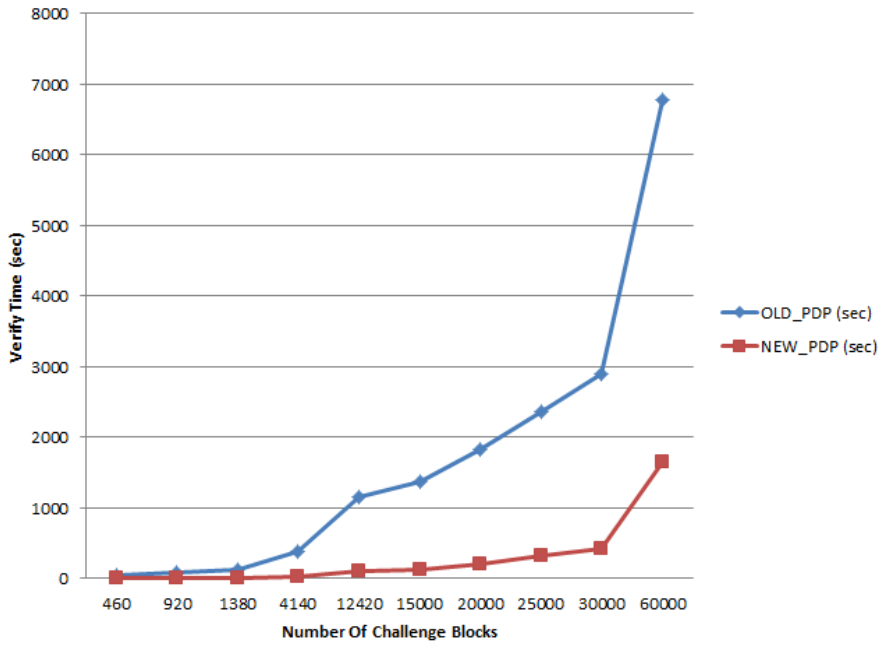

Fig. 8. Challenge Blocks Imapct on Verification Time for 2.1GB File

If we analyze the graphs in **Fig. 7** and **Fig. 8** , we can clearly see the impact on performance of existing schemes. In Fig.7, for 500MB file OLD_PDP scheme is taking around 46mins for 60,000 blocks in challenge request, whereas NEW_PDP scheme is taking around 28mins. But as file is increased to 2.1 GB, performance difference between OLD_PDP and NEW_PDP becomes more clear. In Fig. 8, for 60,000 blocks in challenge request OLD_PDP is taking around 19 hours where as NEW_PDP is taking 30mins. We are assuming backup/archival data having size in GBs and deterministic verification of such big data is practically not possible. Therefore probablisitic verification schemes use sampling of blocks and are more applicable in real scenarios. Since verification time is very high for large datasets, therefore performance is major concern. As our proposed scheme (NEW_PDP) performs comparatively better so it is more applicable in practical scenarios. Data integrity verification remains an active area of research [21]. However, to the best of our knowledge, there exist no such work on data integrity verification with such large data size

As for small size file, even difference between the performance of two schemes is small, however as the file size increases, the proposed scheme outperform the existing scheme. As discussed earlier, the performance degrading factors of existing scheme are variable length of metadata, number of challenged blocks, and the block size chosen. Since block size is fixed so the total number of blocks of file increases with increase in file size. As blocks in challenge are chosen randomly from all available blocks and more effort is required by existing scheme (OLD_PDP) to locate tags by traversing metadata file for higher block indexes. Larger the index number in challenge, more work has to be done to locate the tag in metadata file and for each index, the scheme has to traverse the file from start. The verification time increases for same number of blocks in challenge if total number of blocks increases. Performance of existing scheme degrades with an increase in challenged blocks. The proposed scheme is not affected with increase in file size as the time to locate data and tag is almost constant and are directly accessible by the absolute path in JAR. Even with increase of number of blocks in challenge, the verification time of NEW_PDP is 50-60 times faster than OLD_PDP with larger files. So performance of proposed scheme is not affected by variable length of metadata and unordered indexes in challenge. Increase in number of blocks with increase in size of file, does not impact the search time of block and respective tags.

By analyzing the trend in Fig. 5 for both schemes, one can see that OLD_PDP scheme is not practically usable for large data archives. For example, as graph shows that OLD_PDP scheme takes approximately 7 minutes for 21GB file and NEW_PDP took 7 seconds to verify same file. Conclusion is that OLD_PDP scheme is not scalable whereas NEW_PDP scheme is scalable and will allow us to get timely informed if any corruption occurs.

Secondly, NEW_PDP scheme can provide a stronger guarantee of possession of proof by increasing the number of blocks in request and its performance is still better than existing scheme. This capability makes the proposed scheme applicable for new areas like tamper evident logging as no probabilistic scheme based on sampling exists for logs tampering identification.

4.3 Preprocessing Time Analysis

In OLD_PDP, before placing the data over cloud, preprocessing is done to generate metadata tags. These tags are placed with data over the cloud and storage server. Later on, these tags are used to generate proof of possession. Preprocessing is done just one time for each file on data owner side. Preprocessing of data in proposed scheme is different than OLD_PDP scheme,

therefore preprocessing time of both schemes is compared to analyze the overhead introduced by NEW_PDP. **Fig. 9** shows the preprocessing graph line trend for both schemes with increase in size of input file. The preprocessing time is calculated for both schemes using a data block size of 4KB and metadata tags are of variable length.
Preprocessing is done on data owner side, using Ubuntu 12.04 (OS) with 3GB RAM, 80GB HDD and 2 Processors (2 cores each).

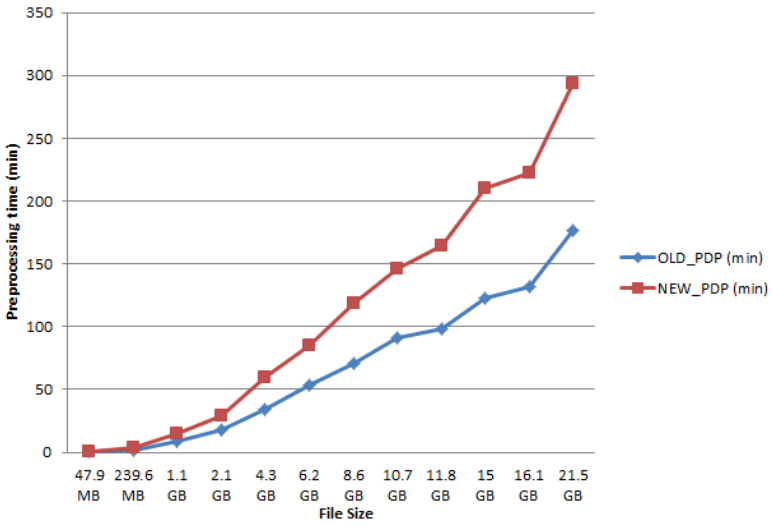

Fig. 9. Scheme's Preprocessing Time Comparison

Proposed scheme is efficient and cost effective but requires more preprocessing time. **Fig. 9** shows the difference in preprocessing computation time. Proposed scheme takes twice the time of OLD_PDP but the overhead is only one time i.e; before placing data over the cloud and is on data owner side. One possible way to minimize this preprocessing overhead is to increase data block size from 4KB to 8KB or even higher as this will reduce the total number of blocks of a file.
Smaller the data block size we chose, greater will be the number of blocks. **Fig. 10** shows that as number of blocks increases with increase in file size, additional work of zipping the blocks and tags is taking much time in NEW_PDP. However, actual reason is that when zip files are added in JAR, as soon as number of entries increases from 1000, time for each new JAR entry starts taking more time and it keeps on increasing. After 100,000 entries in single directory in JAR, impact of adding new entry gets worst because name of new entry is verified with all existing entries to ensure uniqueness. This trend continues up to the completion of preprocessing phase, this additional time of individual entries collectively make up big difference. This preprocessing time can be improved using larger block size (greater than 4KB) because it will decrease the total number of blocks. Therefore, we have less JAR entries and this additional time of adding zip files in JAR will have less impact on preprocessing time.
To show the impact of block size on preprocessing time and output size, we have executed tests by keeping file size same and just increased the block size gradually. File size we used is 10.7 GB.

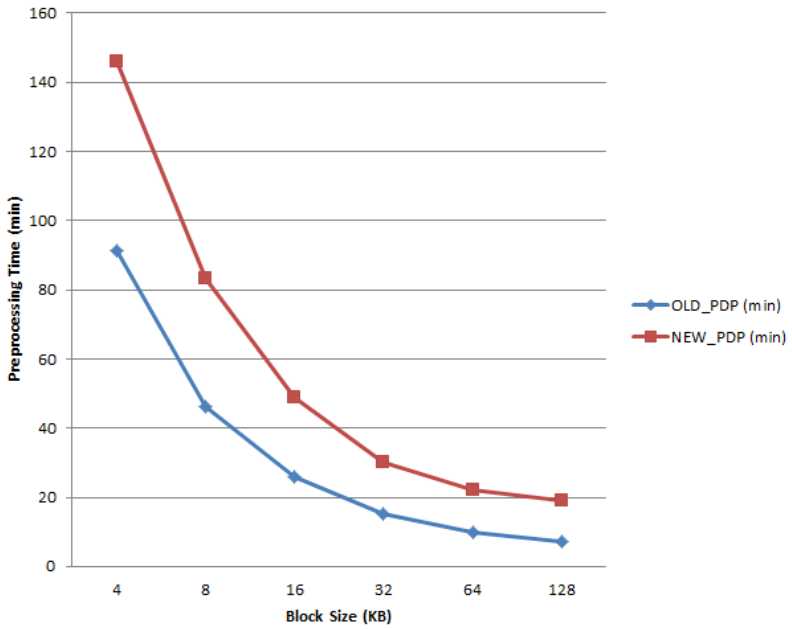

Fig. 10. Block Size Impact on Preprocessing

By analyzing the **Fig. 9** and **Fig. 10**, it is concluded that preprocessing time can be reduced down by choosing the appropriate block size. But even in worst case, preprocessing time is affordable as it is one time activity. Data owner needs to preprocess files only once before outsourcing it to cloud storage. Time complexity is more important for verification process than preprocessing phase because verification is performed on regular intervals and is on going process, whereas preprocessing is done only once. This makes our proposed scheme more applicable for practical scenarios.

We have also plotted the graph of output size in **Fig. 11** to compare the both schemes and to see the impact of increasing block size on output size.
By analyzing the results of all experiments performed based on performance degrading factors, we can see the strength of both schemes. Existing PDP scheme is efficient in block access, communication and computation for small size files but the performance degrades and communication cost increases as file size increases. Main factor of performance degradation of existing scheme is variable length of metadata. Other parameters like block size, unordered collection of block indexes in challenge and number of blocks in challenge increase the impact on verification time because of the variable length of metadata. Variable length of metadata, unordered block indexes in challenge and number of blocks in challenge does not degrade the performance of proposed scheme over large files. The block and metadata tags are directly accessible without any computation, which improves the block access cost and verification time is improved. Since time is constant to access any block and tag, unordered indexes in block also has no impact on performance. As far as communication and storage cost are concerned, results clear show the decrease in size of output in proposed scheme which provides the storage efficiency.

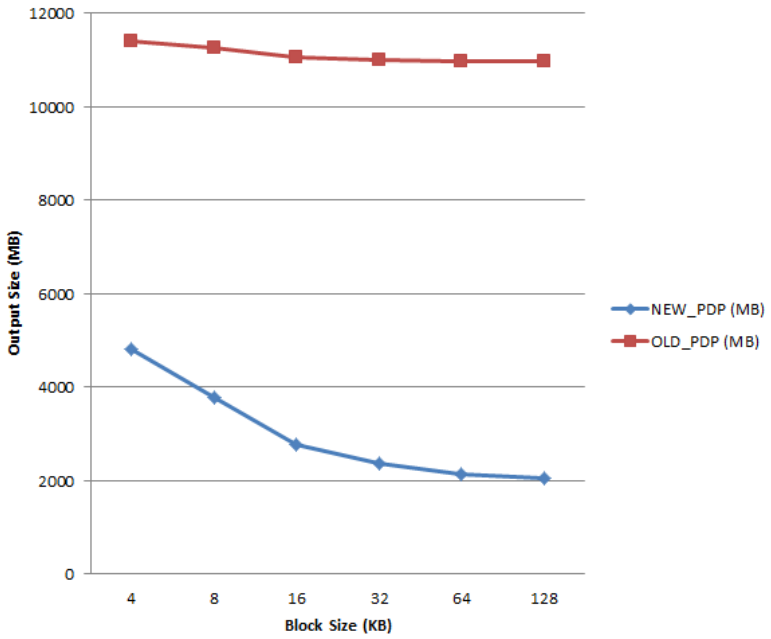

Fig. 11. Block Size Impact on Output Size

5. Conclusion

Provable Data Possession (PDP) scheme is the first probabilistic data integrity verification technique. It depends on homomorphic tags generated against original data blocks for proof generation. Our proposed solution relies on same HVT construction as that of PDP. We have made existing scheme more efficient and workable for large archives without changing the internal mechanism of tags generation and data integrity verification. In our approach, before outsourcing data we divide a single file into small sub files and generate the corresponding homomorphic tags for each sub file. We zip both data file and tags into a single zip file and add zip files to JAR. At the end of preprocessing, we have a single JAR file along with executable code, which can provide the proof of integrity. The compact JAR file is transferred to cloud storage instead of original data file. Our experiment results showed that we are able to achieve about 50% reduction in size of data and respective metadata collectively. We also achieved significant performance gain of 50 to 60 times in computation of proofs for large data files by reducing the metadata search from $O(n)$ *to* $O(1)$. Experimental evaluation of proposed PDP scheme clearly shows that it is scalable, efficient and cost effective for large data archives. Preprocessing phase is time consuming that can be improved, and even in its worst case it is still acceptable as this is onetime activity and is done on data owner side. Storage, communication and computation efficiency of proposed scheme over cloud is remarkable and makes it applicable for real world archival data.

Acknowledgement

The authors would like to pay their gratitude to the anonymous reviewers for providing detailed and useful comments, which helped us a lot to improve the quality of our work. We are also thankful to Higher education commission(HEC) of Pakistan.

References

[1] Ateniese, G.; Burns, R.; Curtmola, R.; Herring, J.; Kissner, L.; Peterson, Z. & Song, D., "Provable Data Possession at Untrusted Stores," in *Proc. of the 14th ACM Conference on Computer and Communications Security*, pp. 598—609, 2007. Article (CrossRef Link)

[2] Ateniese, G.; Di Pietro, R.; Mancini, L. V. & Tsudik, G. (2008), "Scalable and Efficient Provable Data Possession", in *Proc. of the 4th International Conference on Security and Privacy in Communication Netowrk*, pp. 9:1--9:10. Article (CrossRef Link)

[3] Curtmola, R.; Khan, O.; Burns, R. & Ateniese, G., "MR-PDP: Multiple-Replica Provable Data Possession," in *Proc. of Distributed Computing Systems, ICDCS '08. The 28th International Conference on'*, pp. 411-420, 2008. Article (CrossRef Link)

[4] Shacham, H. & Waters, B. (2008), "Compact Proofs of Retrievability," in *Proc. of* Josef Pieprzyk, ed., *Advances in Cryptology - ASIACRYPT 2008, Springer Berlin Heidelberg*, pp. 90-107. Article (CrossRef Link)

[5] Erway, C.; Küpçü, A.; Papamanthou, C. & Tamassia, R. (2009), "Dynamic Provable Data Possession," in *Proc. of the 16th ACM Conference on Computer and Communications Security*, ACM, New York, NY, USA, pp. 213—222. Article (CrossRef Link)

[6] Ateniese, G.; Kamara, S. & Katz, J., "Proofs of Storage from Homomorphic Identification Protocols," in *Proc. of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology'*, *Springer-Verlag, Berlin*, Heidelberg, pp. 319—333, 2009. Article (CrossRef Link)

[7] Bowers, K. D.; Juels, A. & Oprea, A., "HAIL: A High-availability and Integrity Layer for Cloud Storage," in *Proc. of the 16th ACM Conference on Computer and Communications Security*, pp. 187—198, 2009 Article (CrossRef Link)

[8] Bowers, K. D.; Juels, A. & Oprea, A., "Proofs of Retrievability: Theory and Implementation," in *Proc. of the 2009 ACM Workshop on Cloud Computing Security*, pp. 43—54, 2009. Article (CrossRef Link)

[9] Wang, Q.; Wang, C.; Li, J.; Ren, K. & Lou, W., "Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing," in *Proc. of the 14th European Conference on Research in Computer Security*, *Springer-Verlag, Berlin, Heidelberg*, pp. 355—370, 2009. Article (CrossRef Link)

[10] Wang, C.; Wang, Q.; Ren, K. & Lou, W., "Privacy-preserving Public Auditing for Data Storage Security in Cloud Computing," in *Proc. of the 29th Conference on Information Communications'*, *IEEE Press*, pp. 525—533, 2010. Article (CrossRef Link)

[11] Zhu, Y.; Wang, H.; Hu, Z.; Ahn, G.-J.; Hu, H. & Yau, S. S., "Efficient Provable Data Possession for Hybrid Clouds," in *Proc. of the 17th ACM Conference on Computer and Communications Security*, ACM, New York, NY, USA, pp. 756—758, 2010. Article (CrossRef Link)

[12] Ateniese, G.; Burns, R.; Curtmola, R.; Herring, J.; Khan, O.; Kissner, L.; Peterson, Z. & Song, D., "Remote Data Checking Using Provable Data Possession," *ACM Trans. Inf. Syst. Secur.* 14(1), 12:1--12:34, 2011.Article (CrossRef Link)

[13] Lee, N.-Y. & Chang, Y.-K., "Hybrid Provable Data Possession at Untrusted Stores in Cloud Computing," in *Proc. of the IEEE 17th International Conference on Parallel and Distributed Systems*, IEEE Computer Society, pp. 638—645, 2011. Article (CrossRef Link)

[14] Chen, B. & Curtmola, R. (2012),"Robust Dynamic Provable Data Possession," in *Proc. of Distributed Computing Systems Workshops (ICDCSW), 32nd International Conference on*, pp. 515-525, 2012. Article (CrossRef Link)

[15] Krzywiecki & Kuty, M., "Proof of Possession for Cloud Storage via Lagrangian Interpolation Techniques," in *Proc. of the 6th International Conference on Network and System Security*, Springer-Verlag, Berlin, Heidelberg, pp. 305—319, 2012. Article (CrossRef Link)

[16] Mohan, A. & Katti, R., "Provable Data Possession Using Sigma-protocols," in *Proc. of Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE 11th International Conference on*, pp. 565-572, 2012. Article (CrossRef Link)

[17] Sundareswaran, S.; Squicciarini, A. & Lin, D., "Ensuring Distributed Accountability for Data Sharing in the Cloud," *Dependable and Secure Computing, IEEE Transactions on* 9(4), 556-568, 2012. Article (CrossRef Link)
[18] Zheng, Q. & Xu, S., "Secure and Efficient Proof of Storage with Deduplication," in *Proc. of the Second ACM Conference on Data and Application Security and Privacy*, pp. 1—12, 2012. Article (CrossRef Link)
[19] Zhu, Y.; Hu, H.; Ahn, G.-J. & Yu, M., "Cooperative Provable Data Possession for Integrity Verification in Multicloud Storage," *Parallel and Distributed Systems, IEEE Transactions on* 23(12), 2231-2244, 2012. Article (CrossRef Link)
[20] Chen, L., "Using Algebraic Signatures to Check Data Possession in Cloud Storage," *Future Gener. Comput. Syst.* 29(7), 1709—1715, 2013 Article (CrossRef Link)
[21] Esiner, E.; Kachkeev, A.; Braunfeld, S.; Küpçü, A. & Цzkasap, Ц., "FlexDPDP: FlexList-based Optimized Dynamic Provable Data Possession," *Cryptology ePrint Archive*, Report 2013/645, Article (CrossRef Link)
[22] Gondree, M. & Peterson, Z. N., "Geolocation of Data in the Cloud," in *Proc. of the Third ACM Conference on Data and Application Security and Privacy*, ACM, New York, NY, USA, pp. 25—36, 2013. Article (CrossRef Link)
[23] Wang, H., "Proxy Provable Data Possession in Public Clouds, *Services Computing," IEEE Transactions on* 6(4), 551-559, 2013. Article (CrossRef Link)
[24] Zhang, Y. & Blanton, M., "Efficient Dynamic Provable Possession of Remote Data via Balanced Update Trees," in *Proc. of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ACM, New York, NY, USA, pp. 183—194, 2013. Article (CrossRef Link)
[25] Wei, L.; Zhu, H.; Cao, Z.; Dong, X.; Jia, W.; Chen, Y. & Vasilakos, A. V., "Security and Privacy for Storage and Computation in Cloud Computing," *Inf. Sci.* 258, 371—386, 2014. Article (CrossRef Link)
[26] Wang, H. & Zhang, Y, "On the Knowledge Soundness of a Cooperative Provable Data Possession Scheme in Multicloud Storage," *Parallel and Distributed Systems, IEEE Transactions on* 25(1), 264-267, 2014. Article (CrossRef Link)
[27] Wei, L.; Zhu, H.; Cao, Z.; Dong, X.; Jia, W.; Chen, Y. & Vasilakos, A. V., "Security and Privacy for Storage and Computation in Cloud Computing," *Inf. Sci.* 258, 371—386, 2014. Article (CrossRef Link)
[28] Albeshri, A.; Boyd, C. & Nieto, J., "Enhanced GeoProof: improved geographic assurance for data in the cloud," *International Journal of Information Security* 13(2), 191-198, 2014. Article (CrossRef Link)
[29] Fu, D. L.; Peng, X. G. & Yang, Y. L. (2014), "Trusted Validation for Geolocation of Cloud Data," *The Computer Journal*, 2014 Article (CrossRef Link)

Faheem Zafar is a PhD student in Computer Science Department, CIIT Islamabad. He has over 10 years of software development experience. He did his MS from CIIT Islamabad. His areas of research are security & privacy issues cloud computing, secure provenance and performance tuning.

Abid Khan is working as an assistant professor at Computer science department in CIIT, Islamabad. He was a postdoc fellow at Politecnico de Torino, Italy from 2009-2011. He did his PhD from Harbin Institute of technology, Harbin, P.R. China. His research interest includes cryptography, security & privacy issues in distributed computing.

Mansoor Ahmed is working as an assistant professor in in Computer Science Department, CIIT Islamabad. He did his PhD from Vienna Institute of technology. From 2010-2011 he was a postdoc fellow at Indiana University. His research interest includes semantic web, information security and privacy.

Majid Iqbal Khan received his undergraduate degree with majors in Mathematics and Physics (2001) and Master degrees in Software Engineering (2004). He obtained a PhD in Wireless Sensor Networks from University of Vienna, Austria in 2009. He joined COMSATS Institute of Information Technology in 2009, where he is currently an Assistant Professor. His research interest includes wireless sensor networks, MANETS and Internet of things.

Farhana Jabeen is currently working as Assistant Professor in the Computer Science department of COMSATS Institute of Information Technology, Islamabad. She obtained MS degree from EME college NUST (Pakistan) in 2006 and Ph.D. degree from The University of Manchester, UK in 2011. Since then she has been actively involved in wireless sensor networks research. Her other research interests include query processing in wireless sensor network, distributed spatial analysis, distributed computing using ad hoc wireless networks.

Zara Hamid did her PhD from National University of Sciences and Technology (NUST), specializing in the area of Wireless Sensor Networks. She is interested in the design and performance evaluation of communication protocols for wireless ad hoc and sensor networks. She is also interested in Information Centric Networks.

Faisal Bashir is an Associate Professor of Computer Science at Bahria University, Islamabad. He did his PhD from Dokuz Eylul University, Izmir, Turkey. His research interests include QOS issues Wireless Network and Wireless Personal Area Network.

Naveed Ahmad has done his Ph.D. in Engineering Change Management (Process Management) from The University of Cambridge. His research interests include modelling and simulating change/business processes, software engineering for mobile applications.