TALLINN UNIVERSITY OF TECHNOLOGY
Department of Computer Science
TUT Center for Digital Forensics and Cyber Security

ITC70LT
Gvantsa Grigolia 144965

EVALUATION OF DATA OWNERSHIP SOLUTIONS IN REMOTE STORAGE.

Master Thesis

Supervisor: Ahto Buldas
Professor

Tallinn 2016

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käsolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Autor: [Gvantsa Grigolia]

[January 2, 2017]

Annotatsioon

Pilevatalletusteenuste andjad kasutavad andmeedastus ja -talletuskulude vähendamiseks ainutalletuse (dedubleerimise) tehnoloogiat, mis on küll tulus, kuid mille turvanõrkused võivad viia potentsiaalsete rünneteni, mis ohustavad andmete konfidentsiaalsust ja isikute privaatsust. Pärast esimeste rünnete ilmsikstulekut on viimaste aastate jooksul välja pakutud mitmeid lahendusi tagamaks turvalist dedubleerimist. Selle töö eesmärk on neist lahendustest ülevaate andmine, nende funktsionaalsuse iseloomustamine ja uurimine hindamaks lahenduste turvalisust, side- ja talletusvajadusi. Töös vaadeldakse ja võrreldakse seitset turvaprotokolli, lähtudes avalikus teadus- jm kirjanduses kättesaadavat informatsiooni, ja võttes nende majandusliku efektiivsuse hinnangutel aluseks Amazoni veebiteenuste hinnakirja.

Abstract

Cloud storage providers use Data Deduplication technology to reduce the cost of storing and transferring the data. Thought it is a beneficial technology, there are security drawbacks leading to potential attacks. Several solutions were published during the last years, offering a secure implementation of Data Deduplication in the remote storage. The aim of this study is to cover those solutions, learn their functionality and characteristics in order to evaluate the security features and their bandwidth and disk space consumption. The study covers the seven security protocols and provides the results of their comparison from security and cost efficiency perspectives. To evaluate the efficiency, it computes generated data for each solution and applies Amazon Web Services price list to calculate the cost. The security evaluation is based on the solutions' security characteristics provided in the paper works.

Contents

List of Figures

List of Tables

1. Introduction

The cloud computing gained a lot of popularity and the reason is its favorable features for business development. Services that the cloud providers are offering are mostly divided in three categories: Infrastructure as a Service (IaaS) – provides the customers with the outsource hardware solutions, Platform as a Service (PaaS) – provides an environment for developers to implement and deploy their solutions, and Software as a Service (SaaS) – offers the solutions deployed over the web to the end users. The charming characteristics of the offered services are the elasticity and on-demand payment. The user can easily increase or decrease the amount of resources he is using and pay according to the consumption.

As the popularity of cloud computing grows, also the amount of customers' generated data increases. But it turns out that customers sometimes generate identical data.[8] As a result the redundant data is stored on the disk and transferred multiple times over the network. To increase the efficiency of disk space and bandwidth consumption, the Data Deduplication technology was introduced. The idea behind this technology is simple – store data on the disk if it is uploaded first time, but reject all the other attempts to upload the same data and create a reference to the existing data instead. In order to identify the duplication, we calculate the hash digest of the content and compare it with the hashes of existing contents on the disk. The next section covers Data Deduplication concept in more details.

The Data Deduplication is an elegant approach, but it has several drawbacks in terms of privacy and confidentiality. Adversary has possibility to learn the content of the customer's data, when cloud providers use Data Deduplication technology. Data privacy issue is one of the aspects that could break the trust of the customers towards the service providers. So those who want to stay in the market, should build the systems, which takes into consideration privacy and confidentiality issues. To avoid the privacy and data confidentiality attacks in remote storage, there are several solutions, which offer secure implementation of Data Deduplication in remote storage. The secure implementation means that the probability that the adversary learns the content of user's private data is negligible and depends on the security parameter (it is tunable parameter). However there are cases when the overhead of protocols/solutions, in order to hold desired security level is high and not worthy to implement. In this work we go through the seven different solutions (some of them are related as well to each other or are based on the outcomes form previews papers.) and provide the detail description of their functionality, security and efficiency features. The objective of this thesis is to gather such solutions, understand their functionality, compare their security characteristics and provide the operational cost analyses and comparison.

To understand the operational part of the secure implement of the Data Deduplication in remote storage, is an important step before diving into the efficiency and security characteristics comparison. The solutions' security characteristics includes the resilience to different type of attacks, soundness of the protocol developed in solutions, the distribution of input file – weather the given soundness is for only a specific class of distribution or it holds for arbitrary distribution and finally it includes the leakage resilience – if the soundness still holds if the adversary has compliances and can learn some information about file. The cost analyses are based on bandwidth and disk space costs necessary to operate the offered solutions. Based on detailed solution description it is possible to calculate the generated extra data transfer and storage. After obtaining the generated data amount it is easy to apply the AWS prices[1] to get the final extra cost for each solution.

There are following sections included in this thesis: Terms and definition, Background and Related Work, Approach, Evaluation and Conclusions. The Terms and definition is self explanatory section and it provides the description of the terms and concepts that is used in the thesis in order to deliver the objectives. The Background and Related Work covers in main concept of the Data Deduplication technology, how it works and its types. After explaining the concept of the Data Deduplication the section demonstrates the several attacks exploiting the Data Deduplication in remote storage. The Approach section provides a detailed description of the seven solutions. This part is important in order to understand how the protocols function and deliver security. The Evaluation – compares the security characteristics and the cost of solutions. The evaluation is based on the paper works and the implementation of the solutions is out of the scope of the thesis. And at the end we have the Conclusions section, which provides a short summary of contribution.

---

[1] `https://aws.amazon.com/s3/pricing/`, last seen October 3, 2016

2. Terms and Definition

This section provides the list of the terms frequently used in the thesis.

Remote Storage	The disk space, provided by the cloud service providers.
Data Deduplication	Technology to reduce the amount of data.
Client	The client side software used by users to connect the remote storage.
Server	The server side software that runs on server side, provides the data storage environment.
Prover	The participant of the protocol that provides the respond on the challenge. In this thesis the client is refereed as the prover.
Verifier	The participant of the protocol that provides the challenge and verifies the respond. In this thesis the server is refereed as the verifier.
Hash	The string of fixed length that uniquely identifies the data. It is the output of the special class of the functions, called hash functions that has special properties and are widely used in cryptography.
PoW	Proof of Ownership – Interactive protocol to implement secure Data Deduplication in remote storage.

3. Background and Related Work

3.1. Data Deduplication

Cloud computing is an on-demand service. Customers are charged based on used storage and bandwidth.[2] Both service providers and customers are interested in cost efficient solutions of a cloud storage. Data Deduplication offers disk and bandwidth savings. Idea is simple – avoid or remove a duplicated data. This section covers basic concepts of deduplication technology. It lists various methods and processing types and underlines the approaches used in a cloud storage.

3.1.1. Hash Based Deduplication

To remove or avoid duplicated data, it must be detected first. Hash based Data Deduplication uses the hash values of a file (or data chunk) as a file (or data chunk) identifier. Hashes of files are calculated and then are kept on the server. When the file is uploaded first time, its hash is computed and it is compared with the existing hashes on the server. If there is a match, the file is not stored on the disk (or in case of client-side deduplication, is not transfered at all). Instead, server creates the reference, which points on the already existing file, with the same hash value. If computed hash does not match with any of the hashes, the file together with the hash value is stored on the server.[9]

3.1.2. Types of Deduplication

Data Deduplication differs based on processing methods. If it takes place before the client application transfers the file to the server, it is known as client-side deduplication. If it takes place, after the file is uploaded on the server, it is known as server side

---

[2] "With Amazon S3, you pay only for the storage you actually use. There is no minimum fee and no setup cost. Amazon S3 has three pricing components: storage (per GB per month), data transfer in or out (per GB per month), and requests (per n thousand requests per month)." http://aws.amazon.com/s3/pricing/

deduplication. In client-side deduplication scenario, the client application computes the hash of the file and sends it to the server. If the hash already exists on the server side, client application does not send the file. If no match is found, it means,that the file is unique and client application sends it to the server. On the other hand, if client application directly sends file to the server and server computes the hash after it, it is called server-side deduplication. Both processing methods save storage, but client-side deduplication also reduces bandwidth consumption.[10]

Apart of divers processing methods, Data Deduplication differs in processing levels. There are file and block level of deduplication. Difference between them is intuitive. In case of file level, hash of file is calculated and as a result server stores unique files. In block level scenario, files are divided into blocks(fixed or variable size). Hashes of these blocks are calculated and duplicated data on block level is avoided. [10]

The last concept is, single and cross client Data Deduplication. Single client Data Deduplication removes duplicated data in scope of one user. Duplicated data will be stored on the server, if it belongs to different users. On the other hand, cross client deduplication vanishes the user boundaries and unique data would be shared among the users.[10]

3.1.3. Summary

Cloud storage providers are looking for, the most efficient way to reduce the cost. In cross user client-side deduplication case, file or ”chunks” of file are stored only ones on the disk and users are sharing the data. It reduces the bandwidth consumption dramatically, because the deduplication takes palace on client side, and duplicated files are not uploaded at all.[11] Such reductions is attractive for cloud storage providers, but this technology has some security drawbacks. Next section covers potential attacks exploiting the Cross-User-Client-Side Data Deduplication in cloud storage.

3.2. Confidentiality and Privacy Issues in Remote Storage

Taking into consideration the behavior of the Cross-User-Client-Side Data Deduplication, it is easy to learn some general facts. This section focuses on attacks breaching the confidentiality and privacy of remote storage customers' data, when Cross-User-Client-Side Data Deduplication takes place.

3.2.1. Potential Attacks

Danny Harnik was first who has demonstrated, the potential attacks in remote storage related to Data Deduplication technology.[12] The paper covers three cases: file detection, file content detection and covert channel. The first case shows, how trivial is to learn whether the remote server already contains the particular file or not. Attacker uploads the file and observes the network traffic or the time required to upload the file. If the file already is stored on the server, there is no need to upload it again. Client application sends only the hash of the file to the server. The observer detects, that amount of data is smaller then file's size itself(Size of the hash depends on hash function and is smaller then file size). If file is "big enough", observing the time required to upload file on server, is sufficient to learn, whether the server already contains the file or not. The law enforcement authorities, can use this behavior. Check if storage provider contains the file (e.g. file's content is against the law) and later, they can force remote storage service providers to revile the identity of the file owner.

Data Deduplication technology opens the possibility to guess the file content. The approach is straightforward, attacker just uploads all possible variations of file content and waits for occurrence of deduplication. Once it takes place, attacker learns that such file (file with this content) exist on the server. The trick is that, unlike the dictionary attacks it is not detectable. It is the legitimate way to upload new documents on the server.[12] This type of attack is easy to lunch against the files with small min-entropy. To have batter understanding, refer to the following example. Bob is invited at the event in the cinema. He stores his invitation ticket in the cloud. Alice wants to learn the row and the place of Bob's ticket. She puts the Bob's name on the right place and starts to brute force row and place numbers. Alice generates files with different content and uploads on cloud. Once the deduplication takes place, she will get the desired information.

The Data Deduplication technology could be used to establish covert channel. Precondition for this scenario is, that attacker already have to own the victims machine. In order to exchange one bit information "0" or "1", attacker generates two random files and uploads one of them. If the first file is uploaded the covert channel transfers "0" else it transfers "1" bit. Covert channel can transfers more information, by altering the number of files or the meaning of file.[12]

All above stated attacks demonstrate the side channel effects of Data Deduplication. Attackers exploit the vulnerability, that Data Deduplication is detectable. But later Halevi states that main issue is not the detectability, but using the hash value as a proxy in remote storage.[1] He claims that, to use a hash as a proxy to retrieve the file is vulnerable. Owning a small static piece of the file(e.g. hash of the file) does not necessarily mean owning the entire file. He referees to the Dropship[3] open source project, as a brief example of misusing the storage provider. Dropship turn the remote storage provider into CDN (Content Distribution Network) service. For that time Dropbox[4] was operating based on the cross-user client-side deduplication. The users of Dropship, were able to download the file in their folder, just sending the file's hash for check to the Server. This open source project was considered as the violation of Terms of Service of the company and is not operating anymore. Halevi introduces the Proof of Ownership Protocol, which dramatically reduces the probability of the attacker to retrieve the file, without owning it. Next section covers the detail description of Proof of Ownership Protocol and other solutions offered to substitute the hash as a proxy approach for data ownership in a remote storage.[1]

3.3. Summary

The amount of savings offered by Data Deduplication, depends on data type and content produced by users of such services.[8] In case of office workers as users of remote storage, the benefit from deduplication is high. Office workers use mostly

---

[3] https://github.com/driverdan/dropship - "Instantly transfer files between Dropbox accounts using only their hashes"

[4] https://www.dropbox.com/

identical templates to generate the data and the portion of duplication is high. Applying Data Deduplication technology saves bandwidth and disk space. But same time it rises privacy and confidentiality issues. The major weakness is that, client-side deduplication is detectable and using hash as a proof of ownership is not sufficient. Anyone who possesses the hash value of file, is able to retrieve the file from the server. If the attacker obtains the hash of the file, he can retrieve the file from the server and gain unauthorized access to it.

4. Approach

We have demonstrated importance of Data Deduplication technology for remote storage services. And have determine the root cause of breaching the privacy and confidentiality. This section covers the solutions, which refuse to use the static piece of information (hash of the file) as a proxy and offers alternative ways to prove the ownership of the data. We numerate the solutions from one to seven based on published date and show how it works and what are their security and efficiency characteristics.

4.1. Solution # 1

This subsections covers Proof of Ownership (PoW) protocol, introduced by Halevi.[1]PoW involves two parties: Prover and Verifier. The goal of prover is to convince the verifier, that he "owns" particular file. While the goal of verifier is to check if the affirmation of the prover is true. To accomplish their tasks, verifier uses summary value of file, while prover relies on the file itself. Paper [1] offers three solutions, and the subsection reviews all of them, but covers security and efficiency characteristics only for the last one. Before we move to the solutions, we have to underline two constraints. First, attacker may have compliances which own the file, but the total number of bits that attacker can receive from them must be less then initial min-entropy[5] of file. And second, attacker can not interact with compliances during the proving phase.

---

[5] "The min entropy, in information theory, is the smallest of the Rényi family of entropies, corresponding to the most conservative way of measuring the unpredictability of a set of outcomes, as the negative logarithm of the probability of the most likely outcome." "A random variable X has **min-entropy** k, denoted $H_{\propto}(X) = k$, if $\underset{x}{max} Pr[X = x] = 2^{-k}$ "[13]

4.1.1. Setup

The most secure and less efficient from suggested three solutions, uses erasure code.[6] Form each 90% of bits, it is possible to recover the whole file. After the file is encoded using erasure code, next step is to build the Merkle-Tree[14] on the encoded file. The verifier(server) keeps the root value of the computed tree and the number of leaves. During the proof phase, verifier(server) asks the prover(client) for some number of leaves' values and their sibling paths. The verifier checks if all the provided sibling paths give the valid Merkle-Tree root value. Based on the outcome, client is able to retrieve file or not.

Computing erasure code requires access to the file and in case of large files (the files stored on the secondary storage) it raises communication complexity. To increase the efficiency of the protocol, erasure encoding is substituted with universal hashing[15]. First the file is hashed and then the Merkle-Tree is built on the hash. The hashing serves to reduce the size of file up to some predefined number of bits(max length 64MByte). The second solution is more efficient then first one, but it weakens the security. Security requirement for first solution claims: attacker can not retrieve file from the server, if the min-entropy remained in file after attacker receives the bits from compliances, is bigger then security parameter. Erasure encoding substitution with universal hashing, made changes in security requirement as well. For second solution, security requirement stress that, attacker can convince the verifier to grant access to the file, if it receives some T bits from compliances, which can be less then min-entropy of the file. (e.g.64MByte)

Erasure code and universal hashing solutions, both consider that the input file is taken form an arbitrary distribution. On the other hand, the third solution claims that, in realistic scenarios, the attacker always has some information about file which he desires to extract. Thereof, it is reasonable to relax the security requirement and define it for particular class of distribution. Such relaxation of security requirement gives possibility to modify the protocol and make it more space efficient. In particular instead of working with bit vectors, it is possible to divide file into blocks and operate over the blocks. There are three phases to prepare the input for Merkle-Tree: initializing, reducing and mixing. First the M bit size file is divided into m blocks.

---

[6] "The basic premise of erasure coding goes as follows: Take a file and split into k pieces and encode into n pieces. Now, any k pieces can be used to get back the file"

In the initializing phase, l blocks of buffer and IV (Initial Vector) are allocated. Next comes reduction phase, which is a liner mapping. It maps, original file's m blocks to the allocated l buffer blocks. Each block of the file is XORed in specific number in some locations. And locations are taken from IV, which is generated as SHA256(IV[i-1],File[i]). Where i is the block number of the file and IV[0] is defined as SHA256-IV.[7]. The same operations take palace at mixing phase. But with one difference, instead of the file blocks, buffer blocks are taken as an input of XORing.

4.1.2. Security And Efficiency

To demonstrate the soundness of the last solution, it is better to view the file from attacker's perspective. Input file in this scenarios is not take form arbitrary distribution, but form some class of distribution. And it is reasonable for real life scenarios, as attacker always know some peace of information(e.g. file format) about the file that he tries to retrieve. M-bit file with k bits of min-entropy, can be represented from attackers perspective as $\vec{f} \leftarrow \vec{w} \cdot A + \vec{b}$, where $\vec{w} \in \{0,1\}^k$ and is chosen randomly, while $A \in \{0,1\}^{k \times M}$ and $\vec{b} \in \{0,1\}^M$ are chosen by attacker(based some knowledge that attacker has). Protocol uses hash function to prepare input for Merkle-Tree, which is linear mapping, $h(\vec{f}) = \vec{f} \cdot C = \vec{w} \cdot AC + \vec{b}C$.[1] Important part in this linear mapping is that the linear code that is generated by AC matrix must have a large minimum distance. And it is possible to achieve as we are choosing matrix C for mapping. The theorem #3 proved in the paper states that the solution is the secure proof of ownership with soundness $\left(\frac{L-d+1}{L}\right)^t$ where L is reduce buffer, t is number of challenges on Merkle-Tree and d is the minimum distance of the linear code generated by AC matrix.

Time efficiency is one of the important features, that characterizes the protocol and influences decision whether to implement it or not. Halevi evaluates the performance of PoW protocol, and compares it with non-secure Data Deduplication and whole file transfer (without Data Deduplication) implementations of remote storage. Overall time protocol requires, is decomposed in three parts: Client, Server and Net-

---

[7] For SHA-256, the initial hash value, H(0), consists of the eight 32-bit words, in hex. These words were obtained by taking the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers.https://tools.ietf.org/html/rfc4634#section-6.2 last seen October 3, 2016

work time.[8] Client time is calculated as the sum of the subtasks client performs and subtasks are: reading file from the disk, computing the SHA256 hash, going through reduction and mixing phases and computing the Merkle-Tree. Server time – time the server needs to check Merkle-Tree authentication signature. And Network time – respectively the time necessary for data generated by prover and verifier to travel via network. Server and Network time consumption is negligible. (E.g. checking 20 sibling paths "costs" $0.6ms$ and data generated for transmission is around $20KB$. In case of $5Mbps$ network the overhead is $0.1ms$. All together the overhead of Server and Network is $0.7ms$). While the main pressure comes on client side. To compare it with insecure implementation of Data Deduplication, PoW on client side adds reduction and mixing phases and Merkle-Tree calculation. As the result of the tests, reduction phase adds less then 28% time over insecure solution. Mixing phase and Merkle-Tree calculation behavior depends on the size of the file. For small size files(less then $64MB$), the time up-growth is 200% , but it stays constant($1158ms$) once the file grows above $64MB$. PoW is also compared with the solution to avoid deduplication and always send a whole file to the server.

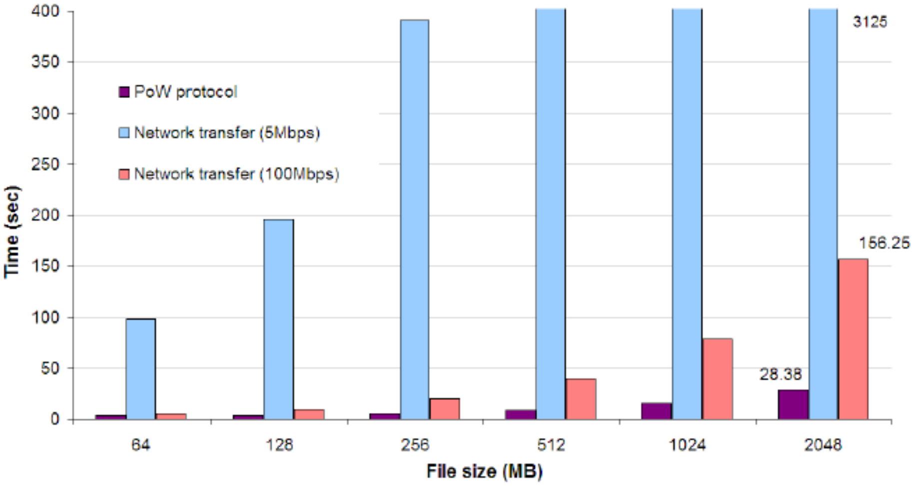

Figure 1. PoW overall performance[1]

.

Protocol is observed in two setups: network with $5Mbps$ and $100Mbps$. The results are following: PoW always consumes less time in $5Mbps$ then transferring the whole file. Once the file grows over the $1GB$ PoW requires 1% time of the file transfer. In

---

[8] "The measurements were performed on an Intel machine with Xeon X5570 CPU running at 2.93GHz. We implemented the protocol in C++ and used the SHA256 implementation from Crypto++"

case of $100Mbps$ network, the protocol has lower bound for file size. For files larger then $64KB$, PoW consumes less time then solution without deduplication . And for files larger then $1GB$, it requires 4% of time of the whole file transfer.

	Dedup Time $= T_d$	File Transfer Time $= T$
PoW	$3.28T_d + 0.7ms$[9] and $1.28T_d + 1.165s$[10]	$0.1T$[11]; $0.4T$[12]; $> T$[13];

Table 1. PoW 1Time Comparison

4.2. Solution # 2

This subsection covers the solution proposed by Di Pietro[2] in his paper "Boosting Efficiency and Security in Proof of Ownership for Deduplication". The motivation of this work is to improve the efficiency of PoW[1] protocol and to avoid the security assumptions that is hard to verify(refereeing to the concept that the file is taken from some class of distribution and not from arbitrary distribution). The subsection includes the scheme description and efficiency analyses in comparison with PoW[1]

4.2.1. Setup

Solution offered by Di Pietro is two party protocol and involves C(Client) as a prover and S(Server) as a verifier. He names a protocol as s-POW. Once S receives the file for the first time it computes the n number of challenges and stores the file on the disk. To compute the challenges S keeps the hash-map data structure $\Im$. It maps files to the tuples and as a key it uses the hash of the file. Tuple contains four elements: ptr – the pointer on the file; $res[]$ – an array of generated challenges–called "responses" (K bit strings); id_c – the highest challenge computed so far ; id_u – number of challenges used so far.

The S uses file digest d (hash of the file), id_c index and server's master key as an

---

[9] For files less then 64Mb: $T_d + 0.28T_d + 2T_d + 0.7ms$

[10] For files more then 64Mb: $T_d + 0.28T_d + 2T_d + 0.7ms$

[11] In 5Mbps network and file size more then 1Gb

[12] In 100Mbps network and file size more then 1Gb

[13] In 5Mpbs for any size of file and In 100Mbps for files larger then 64K

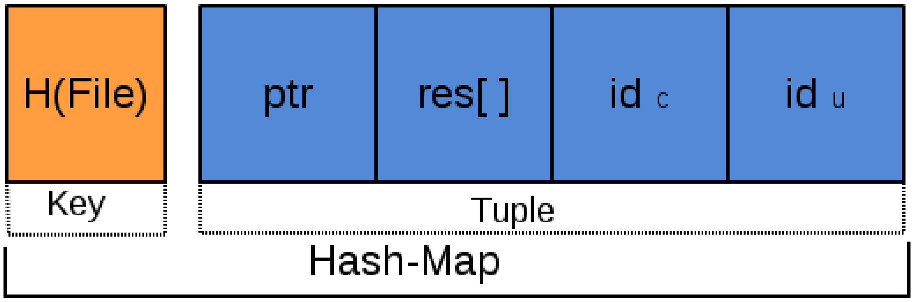

Figure 2. $\Im$ Hash-Map Structure

.

input for the pseudo-random generator F to produce random seed s. The s is an integer and satisfies the following inequality $0 < s < file\_size$. The random seed s serves for calculation of random position in file – represented as bit vector and is unique for each challenge. At the end using the random position and the file as an input, $get - bit$ macros outputs the bit value. Concatenation of such outputs represents the response, which is K bits log and is stored in $res[]$ array. Server computes n number of responses at a time. This approach reduces I/O operations. Computation of responses takes place only, when the client uploads the file, which did not exist on server before or when all the pre-computed responses are depleted.(id_c and id_u control which responses is still valid and how many valid ones are remained.)

If client C attempts to upload the file already located on the server S, S challenges the client. It sends the random seed s and waits for valid response. Client receives s and use same $get-bit$ macro to produce a K bit length response. If received response is the same as pre-computed one, the client succeeds to convince the server.

The s-POW protocol has two other modifications, which are designed to improve efficiency and are more convenient in particular cases. In one case the hash function which is used to calculate the key of the hash-map is substitute with calculation of response(K bit strings) based on the file and the public seed S_{pub}. (This response serves as the key of hash-map) . In second case the file size is used as the key of the hash-map. The second solution is worthy only for large files, as the collision will be extremely high otherwise.

4.2.2. Security And Efficiency

To demonstrate the security of s-POW protocol, Di Pietro shows the probability of adversary to convince the verifier and it is assumed that adversary already owns some large part of file. The Probability of adversary to guess the single bit for the K bit length response is: $P(succ_1) = 1 - \varepsilon(1 - g)$, where ε is fraction of file that is unknown for attacker and g is the probability to guess unknown bit correctly. To convince the server, the adversary should guess the whole K bit length response, and as guessing each bit from K bit vector are independent events, the probability of convincing server is: $P(succ) = (1 - \varepsilon(1 - g))^K$. It means that the success probability of adversary to convince the server depends on K, which is possible to tune based the security requirement. E.g. if the requirement is , $P(succ) \leq 2^{-k}$, where k is a security parameter, then $K = \lceil \frac{k \cdot ln(2)}{\varepsilon(1-g)} \rceil$

Efficiency analyses comprises the CPU computation, I/O in bought client and server side and bandwidth consumption. Di Pietro evaluates his proposed schemes(s-POW and s-POW1 [14]) and compares it with PoW.[15] On client-side both s-POW and s-POW1 schemes are faster then PoW. The complexity up growth of s-POW and PoW schemes are equivalent of file size growth. It is reasonable because in both cases the dominant operation is hashing. While in s-POW1 computation cost becomes constant for large files. As in the s-POW1 no hashing is used and only random disk access is needed to get required bit. For visualization it is better to refer the diagrams provided by authors:

To demonstrate the server-side performance, it is convenient to divide it in two phases: initialization and regular execution. When the file is first uploaded on server-side, that represents initialization phase and all other communication between client and server is the regular execution phase. In initialization phase PoW and s-POW both perform file hashing. It follows with reduction and mixing phase and Merkle-Tree calculation in case of PoW and n challenge computation in s-POW case. In s-POW when all pre-calculated challenges are used, server should calculate them again but this is considered to be a part of regular execution phase. In regular execution phase the PoW performs Merkle-Tree verification while the s-POW per-

---

[14] Modification of s-POW protocol, which uses K bit string instead of file hash as a key in hash-map

[15] "We have run our implementation of both schemes on a 64-bit RedHat box with an Intel Xeon 2.27GHz CPU, 18 GiB of RAM and an IBM 42D0747 7200 RPM SATA hard disk drive.[2]"

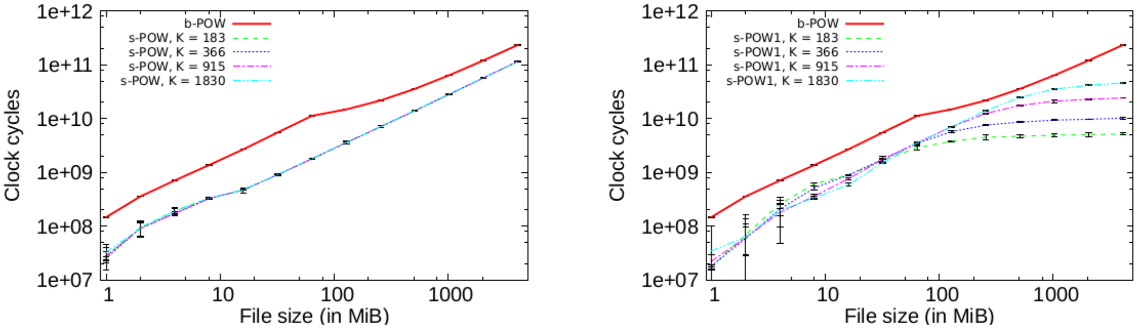

Figure 3. Comparison of running time on client-side [2]

.

forms only look-up to get the correct tuple and the valid response from $res[]$ array. But in addition to look-up, s-POW needs to recalculate the challenges(responses). Dispite the pre-calculated challenges, which are done in order to decrees the file access, on server side PoW is faster then provided s-POW. It is also important to mention the storage efficiency, PoW needs only Merkle-Tree root value to store on server side, while s-POW for operation requires storing of the hash-map data structure. Based on the performed analyses and execution, authors give the asymptotic analyses of schemes.

	PoW	s-POW	s-POW1
Client-side computation	$O(m)hash$	$O(m)hash$	$O(k)PRNG$[16]
Client-side I/O	$O(m)$	$O(m)$	$O(k)$
Server-side computation (initialization phase)	$O(m)hash$	$O(m)hash$	$O(nk)PRNG$
Server-side computation (regular execution phase)	$O(1)$	$O(nk)PRNG$	$O(nk)PRNG$
Server-side I/O (initialization phase)	$O(m)$	$O(m)$	$O(nk)$
Server-side I/O (regular execution phase)	0	$O(nk)$	$O(nk)$
Server-side storage	$O(1)$	$O(nk)$	$O(nk)$
Bandwidth	$O(k\ log\ k)$	$O(k)$	$O(k)$

Table 2. Asymptotic analyses of schemes.POW,s-POW and s-POW1. n is the number of challenges; m is the file size; k is a security parameter. [2]

---

[16] Pseudorandom number generator

4.3. Solution # 3

The subsection reviews the solution offered by Chao Yang in his work "Provable Ownership of File in De-duplication Cloud Storage".[7] Provable Ownership of File, referred as a POF scheme, is two party protocol and helps the client to prove to the server that it indeed owns the file. The subsection first describes the scheme, followed by the security and efficiency analyses. To demonstrate the advantage of their scheme, authors make comparison with PoW protocol.

4.3.1. Setup

POF is a cryptographic protocol that obliges a client to prove to the server that it owns the whole file. The client first sends the hash of the file to the server and if such hash already exist on the server side, the POF protocol invokes and the server challenges the client to prove the file possession. The POF consists of two phases: setup and challenge. In setup phase server decomposes the file F in f blocks. It chooses a random number R_c and generates session key $K_s = h_{sk}(R_c)$ (where sk is pre-shared symmetric key) and two random seeds S_1 and S_2. Those random seeds are used in challenge phase to choose the blocks of the file. The server provides the client with random number R_c, in order to generate the same session key K_s. Client generates the K_s session key and sends back to server the $h_{K_s}(R_c, TS)||TS$ (where TS is the current time stamp) value to confirm the generation of session key. The session key K_s is kept in secret and is used in challenge phase, while the R_c could be deleted.

In challenge phase server sends the c number of blocks($1 \leqslant c \leqslant f$) and two random seeds S_1, S_2 to the client. Random seeds S_1 and S_2 are used to produce block indices i_τ and dynamic coefficient δ_τ where ($1 \leqslant \tau \leqslant c$). Client decomposes the whole file in f block $F = (b_1, b_2, b_3, ..b_f)$ and computes the proof as hash of concatenation of the hashes of choose block and dynamic coefficients.

$$V' = h_{K_s s}(h_{K_s}(b_{i_1}, \delta_1)||h_{K_s}(b_{i_2}, \delta_2)||...||h_{K_s}(b_{i_\tau}, \delta_\tau)) \quad (1)$$

Client sends to the server the generated proof V', server makes the same calculation, but uses the original file and generates the V proof. If $V' = V$ the server is convinced that the client owns the file.

4.3.2. Security And Efficiency

The POF has three major security requirement. First – randomness of indices for the blocks of the file. Second – the original file must be involved for calculation the proof. And third – the calculated proof should be different for different times. When all these three security requirements holds the scheme resistance to cheating is as high as the resistance of collision attack of the hash function used in POF. The authors provide the proof of the following theorem: "For the proposed POF scheme, the complexity for cheating of the ownership verification is at least as difficult as performing strong collision attack of the hash function"

To demonstrate the efficiency of the POF scheme, it is compared with PoW. The authors execute both schemes in the same setup[17] and demonstrate the results for each of them in $milliseconds$. Two protocols computation time is decomposed in three parts: client, server and network computation time.

Client computation time is covered in details. For POF protocol it includes, time to read randomly chosen parts of original file, key derivation time and time to compute the proof. While PoW for simplicity, includes just whole file reading and Merkle-Tree calculation time on original file.(reducing and mixing phases are omitted) And even in this scenario POF is more time efficient on client side. In POF file's portions reading time increases as file size increase, but the time for proof computation could stay the same and does not depend on file size. For PoW – the whole file reading time and Merkle-Tree computation time, both increase as file size grows. Based on results,(see Appendix 1) we can demonstrate the difference in client time computation (in average), between POF and PoW schemes. The schemes had run on different size of files and the file size was doubling each time. The starting size was $0.015625MB$ and it had increased till $1024MB$. We can see that the file size was increased 65536 times. As file size was growing the time consumption was changing. In case of POF for the disk reading , the time was increased approximately 37 times and key derivation and proof computation time was remained mostly constant(around $0.62ms$). So the total time was increased 8 times (at stating point the total time was 0.77 and at

---

[17] The experiments were conducted on an Intel 3.0GHz Intel Core 2 Duo system with 64KB cache, 1333MHz EPCI bus, and 2048MB of RAM. The system runs Ubuntu10.04, kernel version 2.6.34. We used C++ for the implementation. We also used the SHA256 from Crypto++ version 0.9.8b[16]. The files are stored on an ext4 file system on a Seagate Barracuda 7200.7 (ST23250310AS) 250GB Ultra ATA/100 drive.

the ending point – $6.15ms$). In case of PoW – disk reading was increased 28684 times and Merkle-Tree computation approximately 40955 times, overall 39296 time for total time. (at stating point the total time was 0.66 and at the ending point – $25926.39ms$).

Server side computation time is considered to be same or less then client side computation for POF. In case of PoW it is to check the Merkle-Tree authentication signature, which is not time consuming. Network transmitting time, is the time required to transfer the data generated by protocols. In the given setup the data generated for both protocols was less then $1KB$. The server and network times both are negligible and based on the client computation time it was demonstrated that, the POF is more efficient then PoW.

4.4. Solution # 4

The subsection covers solution offered by Jia Xu and his colleagues in the paper work "Weak Leakage-Resilient Client-side Deduplication of Encrypted Data in Cloud Storage".[3] In previous solutions reviewed in this work, the server is considered as a honest player and effort is directed to prevent malicious clients. While this paper addresses both client and server side threats. Paper underlines the importance of confidentiality of user's sensitive data. It claims that, the remote storage provider should not have access to the users sensitive information and proposes the solution of Proof of Work protocol over the encrypted files on client side. Jia Xu expends the security restriction of Halevi's PoW protocol, from specific class of file distribution to arbitrary file distribution. But on the other hand it restricts the data leakage size and security holds if the leakage takes place only before the protocol starts, while in PoW leakage could happen any time, but not during the protocol communication.

Using encryption on client side delivers confidentiality of sensitive information but, on the other hand it rises the risk of *Poison Attack*, also known as *Target Collision attack*.[17] When encrypted file is uploaded on server side, server is not able to check consistency between file and meta-data (e.g. hash of the file). This feature opens possibility to attacker to substituted the encrypted file with the same size malicious one. And if later the owner of the file retrieves it, she gets poisoned file not the original one. Solution demonstrated in this subsection, takes into consideration these type of threats and offers solid security over some restricted leakage conditions.

4.4.1. Setup

The Weak Leakage-Resilient Client-Side Deduplication scheme, is refereed as CSD shortly and is represented with four probabilistic polynomial-time algorithms E, D, P, V . E is an encryption algorithm: $E(F, 1^{\lambda}) \to (\tau, C_0, C_1)$, where F is file, λ is security parameter, τ encryption key and $C_0 \; and \; C_1$ are cipher-texts: C_0 – encrypted key and C_1 – encrypted file. $D(\tau, C_1) \to F$ – decodes C_1 cipher. $P(F) \to y_0, \; y_0 \in \{\tau, \perp\}$ – prover algorithm , which interacts with verifier algorithm and outputs the encryption key τ. $V(C_0) \to (y_1, y_2), \; y_1 \in \{Accept; Rejectg\} \; and \; y_2 \in \{hash(C_1), \perp\}$ – verifier algorithm, which interacts with prover algorithm.

The CSD protocol involves the client and the server. When a client uploads the file first time on the server, it generates the random AES[18] key τ and two ciphers: C_F and C_τ. Where C_F is an encrypted file, using AES encryption and generated key τ and is almost as large as original file. While C_τ is the encrypted random key τ generated by client, using some custom encryption method and the file F as a key. And the size of C_τ is small number – upper bounded by $|\tau|$. After the key and ciphers are generated, client sends hash value of original file F, together with two cipher C_F and C_τ. Server receives the data form client, stores the encrypted file C_F in secondary storage(large but slow storage). While meta-data of encrypted file is stored in primary storage and is represented as key-value pair entry in the lookup database: $(key = hash(F); value = (hash(C_F), C_\tau))$, where $hash(C_F)$ is calculated by server and $hash(F)$ and C_τ is received from client.

If client tries to upload the file – already located on the server side, it first sends the $hash(F)$ to the server. Based on the received $hash(F)$, the server identifies the tuple $(hash(C_F), C_\tau)$ in the lookup database. It retrieves the C_τ value and sends back to client. If the client indeed owns the file, she can decrypt the cipher C_τ using the file as a decryption key and retrieve the ASE key τ. After obtaining the AES key τ, client encrypts the original file, computes the hash of it and sends the $hash(C_F)$ as a proof to the server. Server compares the received $hash(C_F)$ hash value, with the one kept in its database. If the values are equal then the client gains the access to the file, otherwise access is forbidden. After client is identified as honest, she can remove the original file and keep only the ASE key τ, which she will use later for decrypting the downloaded file.

4.4.2. Security And Efficiency

To demonstrate the security of the CSD scheme, authors use the game-based security proof. The security definition states that, adversary can not learn anything new about the a single bit of the file(thereof about the file in whole), from client-side deduplication process, besides the side channel leakage. Security game G_A^{CSD} between challenger and PPT(Probabilistic Polynomial Time) A adversary consists of two learning and two guessing phases. In first learning phase, adversary receives the output of some PPT function y from the challenger. $y < (\varepsilon_0 - \varepsilon_1)$, where ε_0 is a minimum min-entorpy of the file and $(\varepsilon_0 - \varepsilon_1)$ – is a max length of bits adversary is allowed to learn about the file. For it's part $\varepsilon_0 > \varepsilon_1 \geq \lambda$, where λ is a security parameter. Adversary chooses the v indices $(i_1, ...i_v)$ and the challenger chooses the subsequence $\alpha \in \{0,1\}^v$ of the file, such that the each bit chosen based on v indices from that subsequence α corresponds the bit from the file F also chosen based those indices. The challenger chooses the bit $b \in \{0,1\}$ and sets $\alpha_b = \alpha$ and $\alpha_{1-b} \in_R \{0,1\}^v$ and sends those α_0 and α_1 to the adversary. It is followed with the first guessing phase, where an other extractor A_* produces the guess of b bit, $b_{A_*} \in \{0,1\}$. In the second guess phase the adversary tries to guess the b bit, $b_A \in \{0,1\}$.

The CSD is secure in $(\varepsilon_0, \varepsilon_1)$ if the probability of guessing b single bit by extractor A_* plus some negligible in security parameter $negl(\lambda)$, is grater or equal then probability of adversary A guessing the same b bit: $Pr[b_A = b] \leq Pr[b_{A_*} = b] + negl(\lambda)$. Authors construct the secure CSM in the paper and have provide the proof of the security statement in their work.

The CSD scheme efficiency is measured based on running the prover P and the verifier V interactive algorithms.[18]It is compared with the running time of transferring files without deduplication or encryption. The graph provided by authors shows, that it is more efficient to use CSD scheme for secure deduplication rather then avoid it at all and transfer whole files via network.

---

[18] "The test machine is a laptop computer, which is equipped with a 2.5GHz Intel Core 2 Duo mobile CPU (model T9300), a 3GB PC2700-800MHZ RAM and a 7200RPM hard disk. The test machine runs 32 bits version of Gentoo Linux OS with kernel 3.1.10. The file system is EXT4 with 4KB page size" [3]

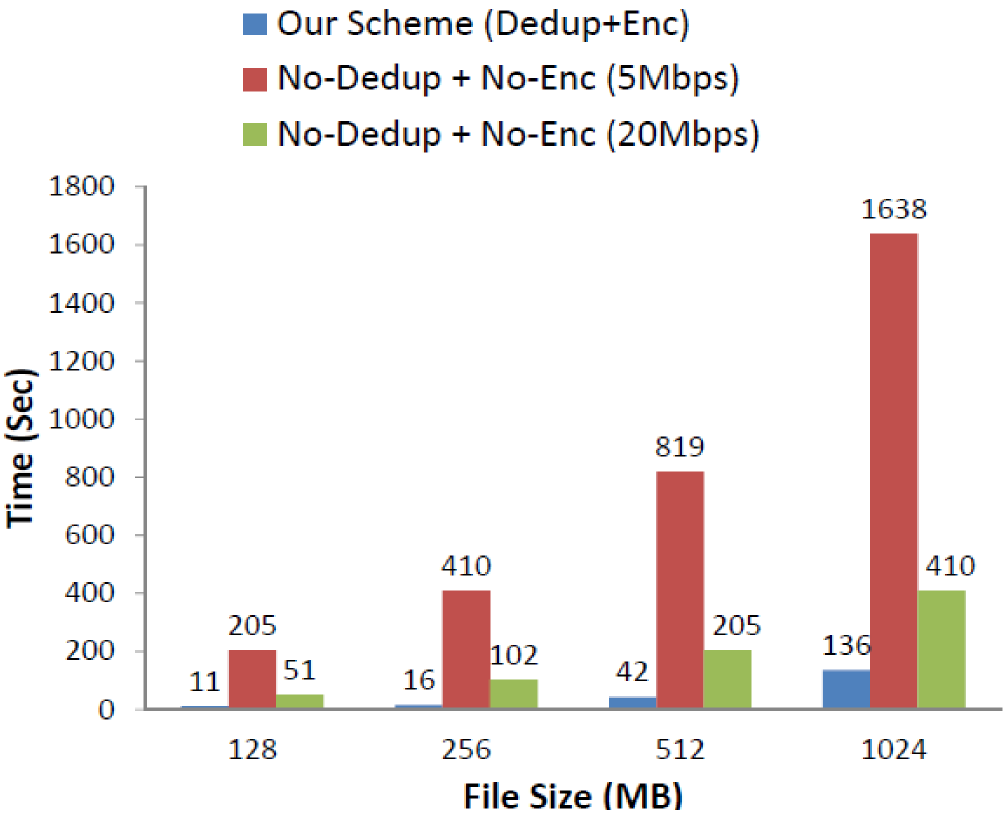

Figure 4. CSD efficiency graph [3]

.

4.5. Solution # 5

The subsection reviews the solution offered by Nesrine Kaaniche and Maryline Laurent in their paper work "A Secure Client Side Deduplication Scheme in Cloud Storage Environments".[4] Like the last solution, it addresses both the malicious clients and the curious-server issues. It provides the means to control the integrity of both players, neither server no client gets fooled. In addition it address the private file shearing issues. The subsection first covers how the solution operates and provides security and efficiency analyses later.

4.5.1. Setup

The security scheme is implemented on OpenStack Swift [19] platform and is based on convergent encryption concept[19], but for identification it uses Merkle-Tree signature. The authors give a clear and simple scheme of cloud storage architecture. When data owner desires to outsource the file F on the remote server, he first encrypts it,

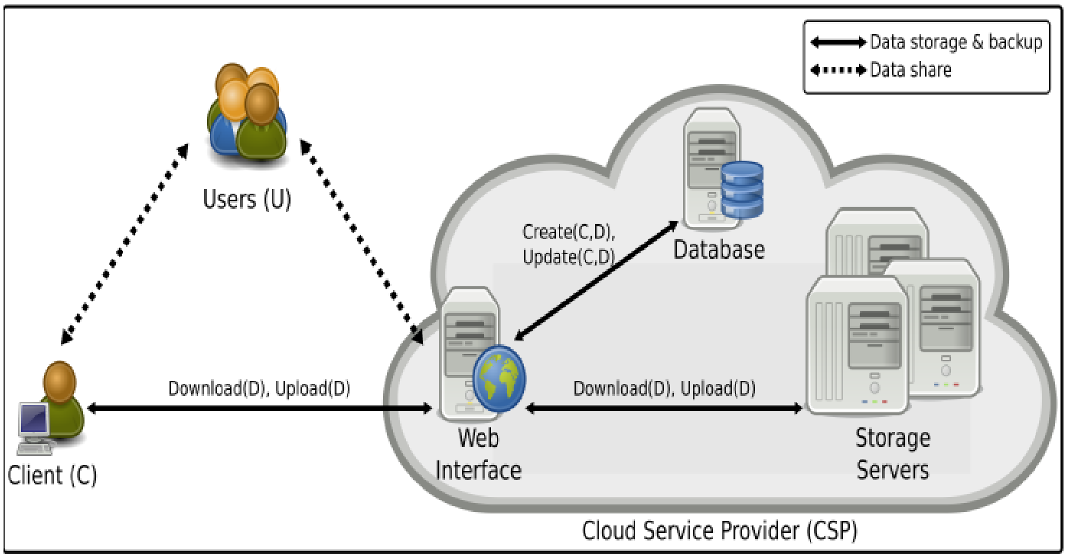

Figure 5. Client uploads and stores the data on server and Users are allowed to access (depend on their access rights) the content of stored data [4]

.

---

[19] The open-source cloud storage project - https://www.swiftstack.com/product/openstack-swift, last seen October 4, 2016

and uses the file content hash K_F as an encryption key. He builds the Merkle-Tree on the encrypted file and uses the root value MT_F as the file identifier, which is unique on the remote storage. Client sends the MT_F identifier[20] to the server and checks whether the file already is located on the server or not. The server checks in the database and if the received identifier does not matches, asks client to upload the file. The client in his turn, sends the encrypted file and the encryption key K_F – encrypted by the public keys of the authorized users (users with whom client desires to share the file). Later the encrypted key K_F is included in the file meta-data. After the server stores the file, it sends the acknowledgement message to client, which includes the stored file's URI(Uniform Resource Identifier).[21]

If the file is already located on the server, then the server sends to the client some number of random indices of Merkle-Tree leaves. The client calculates the sibling paths of chosen leaves and sends (leaves together with sibling paths) as proof of ownership to the server. The server validates the received proof and if it holds, it sends acknowledgment on storing the file and the URI of requested file to the client. Otherwise, client is failed to access the file.

When client desires to access the outsourced data he owns – he sends the file URI to the server. The server, checks in the database, whether the client owns the file. If the client is the owner of the requested file, server sends encrypted file to the client. After receiving the file, client first extracts the file meta-data(i.e. encrypted K_F encryption key), decodes the encrypted K_F key with own private key and use K_F to decode extracted file.

4.5.2. Security And Efficiency

Authors offered very brief overview of security, without any game-based or simulation-based security proof. They address three main issues in their scheme: Data confidentiality, privacy and access control. When client wants to store new data to the remote storage, he calculates the Merkle-Tree over the encrypted file and sends the

---

[20] The client also adds nonce n to the identifier to avoid a replay attack

[21] The solution is implement on Swift, which is an object store, so each file could be reached by URI

root value together with nonce n. Nonce prevents from replay attacks[22](stilling the identifier), while transferring the data. But if the adversary still will be able to obtain the identifier (root value of the Merkle-Tree), he is required to prove the ownership of the file and this proof is based on Merkle-Tree lemma [20] and prevents malicious client to access the confidential data. Scheme also prevents curios servers to reach the users' private data and built users' profile. The server is not able to access the data, because it is encrypted and the encryption key is secured with asymmetric encryption. Access control is managed by embedding the symmetric encryption key K_F encrypted by public keys of the users who are authorized to access the file.

To evaluate the performance of the solution, authors implement the OpenStack swift framework and integrate it with their own scheme for security.[23] The scheme's performance time is decomposed in two parts: Client and Server computation time. The client computation time consists of the data encryption/decryption time and data upload/download time. For encryption/decryption of file the scheme uses symmetric AES(Advanced Encryption Standard) in CBC(Cipher Block Chaining) mode. The authors examine the performance of file encryption/decryption using different key sizes(key size = $128, 192\ or\ 256bits$) of AES encryption and different file input sizes (file size = $k * 10^5$ where $k \in \{1, 2, ..10\}$). The results exposes – the computation time depends on a key size and a file size and as they are increasing the time of computation increases as well. The encryption/decryption time for the smallest key($128bits$) and the smallest input file size($0.1MB$) is 1 ms, while with the largest key size and input file size (key size = $256bits$, file size = $1MB$) is less then 12 ms. The authors examine the upload/download time and the result shows – uploading time is greater then downloading time. Also time remains constant for small files (file size < $5 * 10^4$ bits), while for larger files it depends on file size and time increase as the file size increases. To compare the cryptographic operations to the file transfer operation on client-side, it is easy to see that cryptographic operations are consuming much less time, then file upload/download. And the comparisons that is given in the paper [19] claims, that encrypting of the $0.8MB$ size file takes $0.1ms$, while uploading it takes $10s$, which means the encryption operation is 1% of uploading operation. For securing the AES encryption key, the scheme uses ECC (Elliptic

---

[22] "Replay attacks are the network attacks in which an attacker spies the conversation between the sender and receiver and takes the authenticated information"

[23] "For our tests, we used 1000 samples in order to get our average durations. In addition, we conducted our experiments on an Intel core 2 duo, started on single mode,where each core relies on 800 MHz clock frequency (CPU).[19]"

Curve Cryptography).[21] Unfortunate the paper[19] does not covers the server time consumption and there is no any comparisons with other schemes.

4.6. Solution # 6

This subsection is dedicated to a paper work "A Tunable Proof of Ownership Scheme for Deduplication Using Bloom Filters".[6] The paper offers the data ownership proof scheme based on Bloom Filter.[22] The bloom filter is a time and space efficient data structure. It serves to identify, whether the element is the member of the set or not. The issue with this data structure – it has false positives. The element which does not belongs to the set, could be considered as the member, but never vice-versa. Bloom filter allocates s bits length vector (all bits set to 0) and uses n number of different evenly distributed hash factions. To add the element in bloom filter, the element is hashed with the n different hash functions. The results of hashes is the indecies in the s bits length vector and in those indecies bits are set to 1. To check if the particular element belongs the set, it must be hashed using the n different hash functions to produce the indecies. After the indecies are produced, the vector is examined in those indecies and if all the bits are set to 1, then the element belongs the set.

4.6.1. Setup

The Proof of Ownership Scheme for Deduplication Using Bloom Filters, also referred as bf-POW is the two party protocol. It represents an interaction between C as a client and the S as a server. There are two scenarios, first when uploading the file first time on the server and second trying to upload the file already located on the server. In the first scenario, client sends the hash of the file h_f to the server. The server identifies that the file is unique, based on the fingerprint h_f and asks the client to upload the whole file f and initializes all the required data structures, including bloom filter. After receiving the file f, server divides it into the equal size chunks, then computes the tokens, using some hash function: $H : \{0,1\}^B \rightarrow \{0,1\}^l$, where the B is a chunk size and l is a token size. And finally the indecies are generated form token using some pseudo random function, $PRF : \{0,1\}^l \rightarrow \{0,1\}^n$ where n is a positive integer. Those indecies are inserted in the bloom filter BF data structure.

The server keeps the associative array A, which use the hash of the file h_k as a key and the tuple $\{f, BF, AL\}$ as a value. In the tuple the f is the content of the file, BF represents the bits in the bloom filter and the AL is the list of the client identifiers $id(C)$, who are owning the file. After the file is uploaded first time on the server , the entry is added to the array A.

In the second scenario, when the file is already located on the server, there is no need to upload the file, instead the client must proof that he indeed owns it. The server initializes the array pos, which holds the randomly chosen chunks' indecies and J is the length of the pos array. The server sends pos array to the client and waits for the tokens generated based on pos array. The client performs the same operation as the server, in order to generate the tokens and creates the array res which holds the generated tokens. $res[i] \leftarrow H(f[pos[i]])$ where $0 \leq i < J$ and J is the number of randomly chosen indecies. The client sends res array to server and the server generates the indeces based the PRF and the res array and checks whether each output PRF belongs to bloom filter or not. If all outputs belongs to bloom filter then the client is considered as the file owner and the $id(C)$ is added to the $A[h_f].AL$ list. Otherwise the client fails to prove the file ownership. Later when client request to download the file , the server will check, if the $id(C) \in A[h_f].AL$

4.6.2. Security And Efficiency

To demonstrate the security of the bf-POW scheme, authors evaluate a probability of the adversary $\bar{C}$ convincing the server without owning the file. The adversary is allowed to communicate with file owners and receive some information about the file (but not during while the protocol takes place). The probability that adversary knows the B bits long randomly chosen chunk of the file is p. And the probability to guess the randomly chosen byte is g. In order to pass the proof the adversary $\bar{C}$ should provide J tokens, based on randomly chosen J chunks by the server. And those tokens used as seeds of PRF should produce the indecies which will belong to the bloom filter for the given file f. In order to succeed there are two options. First $\bar{C}$ should produce the correct token and the second – occurrence of false positive when checking for bloom filter membership. The probability of false positive in bloom filter is – p_f. The p_f depends on the size of the bloom filter and the number of hash functions.[24] The probability to produce one randomly chosen token from the J tokens correctly is

$$P(in_i) = P(tok_i) + p_f * P(\bar{tok_i}) \tag{2}$$

where in_i and tok_i are both events: in_i – i-th generated token, is a seed of the index which is the member of bloom filter; tok_i – adversary generates token correctly. In its part probability of event tok_i is decomposed as the probability of knowing the chunk and probability of guessing it. The p is the probability that the adversary knows the i-th chunk. If adversary knows the i-th chunk, it means that he is able to compute the i-th token correctly. If he does not know, he should guess. Probability of guessing B bits long chunk is lower, then probability of guessing l bit token directly, $g^B << 0.5^l$. From adversary perspective better to guess directly a chunk.

$$P(tok_i) = p + (1 - p) * 0.5^l \tag{3}$$

In order to succeed adversary needs to produce J number of tokens form randomly chosen chunks. To compute the probability of success, we use these two formulas : (2) and (3)

$$\begin{aligned} P(succ) &= P(in_i)^J = (P(tok_i) + p_f P(tok_i))^J \\ &= (p + (1 - p) * 0.5^l + p_f * (p + (1 - p) * 0.5^l))^J \\ &= (p + (1 - p) * 0.5^l) + p_f(1 - p)(1 - 0.5^l)^J \\ &= (p + (1 - p)(0.5^l) + p_f(1 - 0.5^l))^J \end{aligned} \tag{4}$$

We can see that the probability of success dependents on J. And it is possible to choose J in such a way that the $P(succ) \leq 2^{-k}$, which is negligible in the security parameter k.

Authors provide the asymmetric analyses and the results of the experiments of bf-POW, in comparison with POW and s-POW schemes. In bf-POW each time the client tries to upload the file, he hashes it. Also the client is required to calculate the tokens, meaning the J times hashing operation over the l length chunks. On server side there are two phases, with different computation complexity in each phase. In initialization phase (when the file is uploaded first time on server side) the dominant cost is

---

[24] If J is a total number of elements that could be inserted in the bloom filter and p_f is the probability of false positives, then the size of bloom filter is $s = \lceil -\frac{N\ ln\ p_f}{(ln2)^2} \rceil$ and the number of independent hash functions is $n = \lceil \frac{s}{N} ln2 \rceil$

the hashing operations used to insert elements in bloom filter. As there are n different functions, it requires n hashing operations. Also in initialization phase server should load the whole file into the memory, while in execution phases (when the file is already located on the server) the server needs only bloom filter to be loaded in the memory, and the indecies are generated by hashing the tokens received from client.(res[] array). In terms of bandwidth efficiency, bf-POW generates J tokens. (the client sends tokens to the server in order to prove ownership). The complexity of the schemes is represented in big-O notation, is provided by authors in the following table :

	PoW	s-POW	bf-POW
Client computation	$O(F)hash$	$O(F)hash$	$O(F)hash$
Client I/O	$O(F)$	$O(F)$	$O(F)$
Server init computation	$O(F)hash$	$O(F)hash$	$O(F)hash$
Server regular computation	$O(1)$	$O(nk)PRF$	$O(\frac{l{*}k{*}(log1/p_f)}{p_f})hash$
Server init I/O	$O(F)$	$O(F)$	$O(F)$
Server regular I/O	$O(0)$	$O(nk)$	$O(0)$
Server memory usage	$O(1)$	$O(nk)$	$O(\frac{log(1/p_f)}{l})$
Bandwidth	$O(k\ log\ k)$	$O(k)$	$O(\frac{lk}{p_f})$

Table 3. Asymptotic analyses of schemes:POW,s-POW and bf-POW. F is the file size; k is a security parameter; n is number of challenges in s-POW; l is a PRF output size; p_f is a probability of false positive in BF [6]

The authors have implemented POW, s-POW and their bf-POW schemes, in order to compare the time performance results to each other. [25] The results show that in bf-POW the better performance depends on the size of token, which in its part also decries bandwidth consumption. Smaller tokens influence the size of bloom filter and increase it in order to keep the security in place.(decries the probability of false positives.) In the worst case bloom filters require $2MB$ additional storage per file. Bf-POW is always faster then s-POW scheme on server side and is the fastest among those three on server side for specific p (p is the probability that adversary knows the randomly chosen chunk of the file). On client side, bf-POW is faster then POW and is a little slower then s-POW.

---

[25] "The benchmarks are run on an Intel Xeon 2.27GHz CPU with 18 GiB of RAM running RHEL Server release Santiago (6.1). The input files contain random data, and their size ranges from 1 MiB to 4 GiB, with the size doubled at each step."

4.7. Solution # 7

"An efficient confidentiality-preserving Proof of Ownership for Deduplication" paper work[5], is the last solution, refereed as the ce-POW, covered in this thesis. The ce-POW scheme addresses the privacy and confidentiality issues in remote storage and considers both malicious client and honest-but-curios server as an adversary. It uses the mix of convergent encryption and proof of ownership protocol, and ensures to avoid poison attack, which is considered to be the drawback of original solution of convergent encryption. Like the previous subsections, this also covers setup of the scheme, describes how it operates and analyses its security and efficiency.

4.7.1. Setup

The ce-POW is two party protocol, involving client C as a prover (each client C has its' own identifier $id(C)$) and the server S as a verifier. The protocol has two phases: initialization and challenge. When client uploads the file first time on the server, it is considered as an initialization phase. Client sends the file size to the server, in order to receive the number of chunks to dived the file. After the client obtains the number N, he divides the file f into N chunks, encrypts each of them using convergent encryption[26], hashes each encrypted chunk $token[i] = H_2(E_{H_2(f[i])f[i]})$ using the $H_2 : \{0,1\}^B \rightarrow \{0,1\}^l$ where B is the chunk size and the l is a token size and $f[i]$ is the i-th chunk of the file. And based on array of $token[]$ generates the fingerprint h_c, using the hash function $H_1 : \{0,1\}^* \rightarrow \{0,1\}^n$. The client sends to the server all the encrypted chunks together with the fingerprint h_c. After server receives the data from the client, based on encrypted chunks it calculates the h_c in order to avoid the poison attack. If the calculate fingerprint is the same as the received one from the client, the server inserts the entry in the associative array A, which exist on server side and takes part in challenge phase. The associative array A uses the fingerprint h_c as a key and $\{ENC, CH, RES, AL\}$ tuple as a value, where $A[h_c].ENC$ stores the encrypted chunks; $A[h_c].CH$ – 10 000 challenges, each represents the J number of chunk indecies; $A[h_c].RES$ – expected responses on the challenges and $A[h_c].AL$ keeps the list of legitimate file owners.

---

[26] CE – convergent Encryption uses the content hash as an encryption key, in this case it use $H_2 : \{0,1\}^B \rightarrow \{0,1\}^l$

If client tries to upload the file already located on the server, it is the challenge phase and client sends the h_c and the server identifies that the file f, with the fingerprint h_c already located on the disk and there is no need to upload it.(If there is no entry in the associative array A, with the key h_c, the initialization phase stars.) Instead the server challenge the client to prove the file ownership. The server picks the first unused challenge – $pos[]$ which is J size in order to hold the randomly chosen J chunks' indecies and sends it to the client. The client picks the indecies form the $pos[]$ array, calculates the $tokens$ based on those indecies and generates the $res[]$ array, which holds i-th token on i-th position. The client sends the array $res[]$ to the server and server compares pre-calculate responses $A[h_c].RES[i]$. If the match is found then the client is added in the list of file owner $id().AL$, otherwise the proof of ownership is failed.

4.7.2. Security And Efficiency

The security requirement of the ce-POW, claims that the scheme is secure if the probability of the adversary convincing the server is negligible in security parameter k , when the adversary does not owns the part of the file, which is larger then some predefined threshold S_{min}.The adversary is allowed to communicate with file owners and receive some information about the file (but only before protocol starts). The probability that adversary knows the B bits of randomly chosen chunk of the file is p. And the probability to guess the randomly chosen byte is g. In order to pass the proof the adversary $\bar{C}$ should provide J tokens, based on randomly chosen J chunks by the server. And the probability of success is:

$$P(succ) = P(tok_i)^J \tag{5}$$

where tok_i is an event that adversary generates i-th token correctly.

The probability of event tok_i is decomposed as the probability of knowing the chunk and probability of guessing it. The p is the probability that the adversary knows the i-th chunk. If adversary knows the i-th chunk, it means that he is able to compute the i-th token correctly. If he does not know, he should guess. Probability of guessing B bits long chunk is lower, then probability of guessing l bit token directly, $g^B << 0.5^l$(g^B – probability of guessing B bits;0.5^1 – probability of guessing l bit token). From adversary perspective better to guess directly a token. Then the probability of guessing the i-th token correctly is:

$$P(tok_i) = p + (1 - p) * 0.5^l \tag{6}$$

Substitute $P(tok_i)$ in formula (5) with formula (6):

$$P(succ) = P(tok_i)^J = (p + (1 - p) * 0.5^l)^J \tag{7}$$

We can see that the probability of success dependents on J. And it is possible to choose J in such a way that the $P(succ) \leq 2^{-k}$, which is negligible in the security parameter k.

To demonstrate the efficiency of the the ce-POW scheme, authors provide the asymptotic analyses and experimental results of the implementation. The ce-POW is compared with s-POW and bf-POW solutions. In client side ce-POW requires encryption and computation of two hashes, while bf-POW computes the hash of the whole file each time it makes a request to the server. On server side in initialization phase, the bf-POW make n number of hash operations in order to insert the element in bloom filter. For ce-POW the cost is two hashing operations. The bandwidth consumption for ce-POW is J [27] tokens that, a client sends to server in order to prove the data ownership. The number of tokens increases as the security parameter k is increasing. The bf-POW has similar bandwidth consumption, in difference that the number of J tokens could be shortened in exchange of rising the false positives in bloom filter. While the s-POW needs only to transfer K bits of randomly chosen from file. Authors provide asymptotic analyses in the table :

Three schemes ce-POW, s-POW and bf-POW was implement, in order to compare their performance results. [28] For ce-POW implementation the parrameters was chosen in following way: security parameter k was set to 66, threshold of leakage $S_{min} = 64MB$, tokens size $l \in \{16, 64, 256, 1024\}$, probability of adversary knowing the chunk of file $p \in \{0.5; 0.75; 0.9; 0.95\}$ and number of requested challenges

---

[27] the implementation has 4 different size of tokens $16, 64, 256$ *and* 1024 *bits*

[28] "The experiments have been performed on a AMD Athlon(tm) II x2 220 processor with 4GB of RAM. Input fles have been randomly generated and their sizes range from 4MB to 2GB doubling the size at each step."

	ce-PoW	s-POW	bf-POW
Client computation	$O(B) * CE * hash * hash$	$O(F)hash$	$O(F)hash$
Client I/O	$O(F)$	$O(F)$	$O(F)$
Server init computation	$O(B) * hash * hash$	$O(F)hash$	$O(F)hash$
Server regular computation	$O(n * l * k) * PRNG$	$O(nk)PRF$	$O(\frac{l*k*(log1/p_f)}{p_f})hash$
Server init I/O	$O(F)$	$O(F)$	$O(F)$
Server regular I/O	$O(0)$	$O(nk)$	$O(0)$
Server memory usage	$O(n * l * k)$	$O(nk)$	$O(\frac{log(1/p_f)}{l})$
Bandwidth	$O(l * k)$	$O(k)$	$O(\frac{lk}{p_f})$

Table 4. Asymptotic analyses of schemes: ce-POW,s-POW and bf-POW. F is the file size; k is a security parameter; n is number of challenges in s-POW; l is a token size; p_f is a probability of false positive in BF [5]

$J \in \{91, 182, 457, 914\}$. The results shows that in initialization phase on server side the fastest scheme is bf-POW. In bf-POW the server initializes and inserts the elements in bloom filter. To insert the element in bloom filter server divides the file in chunks, computes the tokens form chunks which is hashing operation and then computes the indecies , which are inserted in bloom filter. For s-POW, server in initialization phase pre-computes beforehand the $n = 10000$ challenges, using some PRF on randomly chosen J bits. While in ce-POW, server hashes each encrypted chunks and then pre-computes $n = 10000$ challenges. The time that takes ce-POW solution includes the effort to check for poison attacks and defend against the hones-but-curios-server, which is not considered neither in s-POW no in bf-POW solutions. The authors put the result on the chart:

On the client side the picture is the same – bf-POW is the fastest solution while the s-POW is the slowest. The ce-POW is much better then s-POW solution and is comparable with bf-POW, but again it is important to see that only the ce-POW addresses the privacy and confidentiality issues fro both client and server side.

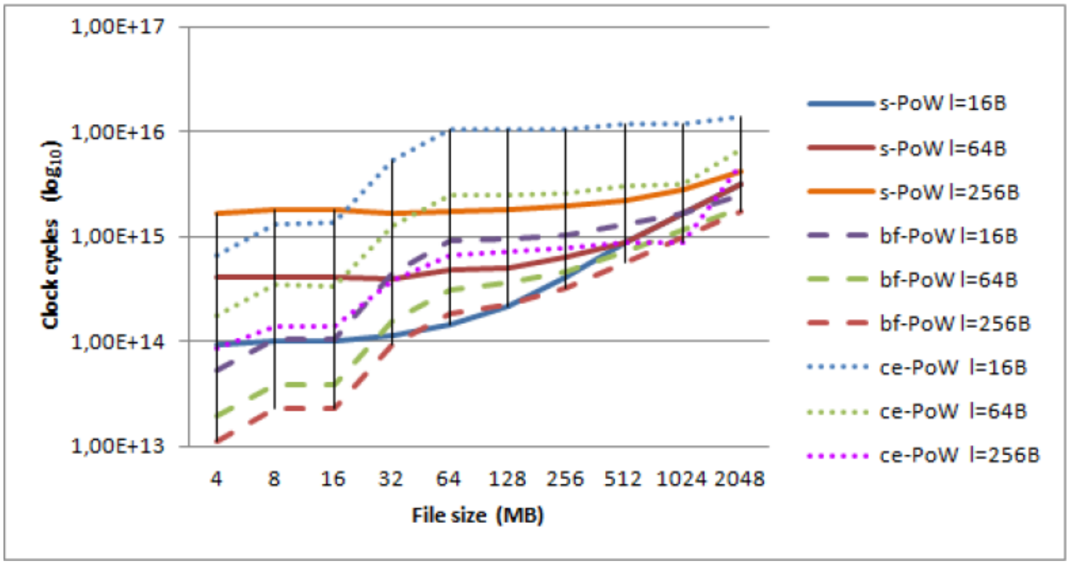

Figure 6. Comparison of the server side initialization phase: ce-POW, s-POW and bf-POW [5]

.

4.8. Summary

The section Approach covered the seven different solutions, which address the privacy and the confidential issues in remote storage during the data deduplication. It is fundamental to understand in details the state-of-art solution, also refereed as PoW introduced by Halevi. He addresses the cause of the problem (using the hash-as-a-proxy) and offers the novel solution to deal with it. All the other solutions covered in previews subsections were inspired by Halevi's work. Some of them(Solutions 4, 5, 7) expend the attack vectors and implement the mix of encryption and proof of ownership, in order to prevent the cloud services providers from profiling the customers' data. Most of them improve the security, by avoiding the restriction of choosing the file from the particular class of distribution, which is hard to prove. Detail understanding of how each scheme works and what are their security and efficiency criteria, helps to move on the next section and evaluate them in comparison with each other.

5. Evaluation

Previews section had covered the details of the existing solutions for proof of data ownership in remote storage during the data deduplication. Implementing of those solutions and evaluate their efficiency in terms of I/O and CPU performance is out of scope of this work. Instead this section addresses a bandwidth and a space consumption, which could be calculated without implementation of the solutions. Based on the results, we can see additional cost that should be payed to maintain the security level. Calculation is based on the AWS prices. [29]. But before the cost analyses, we cover the security aspects of the schemes and evaluate them based on their characteristics described in previews sections.

5.1. Security Evaluation

It is not straightforward to compare the security of the provide solutions. In the most cases the probability of breaking the security of the scheme is negligible in the given security parameter k and k is always tunable. But the diversity follows from the variety of the attacks that the schemes are addressing and also from the restrictions in security definition of the protocols. We have demonstrated five general type of attacks in previews sections: File Detection (the adversary uploads the file and observers, whether the deduplication occurs or not), Content Detection (brute-forcing the content of the file with low min-entropy, detection via deduplication),CDN (using the remote storage as a Content Distribution Network, while it is not meat to be the one), Honest-but-curios-servers(Servers are able to access the private data of the user as it is available in clear text) and Poison Attack (When encrypted file is uploaded on server side, server is not able to check consistency between file and meta-data (e.g. hash of the file)). In the table below it is demonstrated which solutions address to which particular attacks.

Under the security of the scheme/protocol, we imply the probability of the adversary to pass the protocol and convince the verifier. The lower the probability is , more secure is the scheme. In most cases it is demonstrated that the probability of

---

[29] `https://aws.amazon.com/s3/pricing/`, last seen October 3, 2016

[30] the scheme does not uses convergent encryption and the Poison Attack is not applicable.

	File Detection	Content Detection	CDN	Honest-but-curios-servers	Poison Attack
Solution 1 PoW	✓	✓	✓	–	N/A[30]
Solution 2 s-POW	✓	✓	✓	–	N/A
Solution 3 POF	✓	✓	N/A	–	N/A
Solution 4 CSD	✓	✓	✓	✓	✓
Solution 5 OpenStack based	✓	✓	✓	✓	✓
Solution 6 bf-POW	✓	✓	✓	–	N/A
Solution 7 ce-POW	✓	✓	✓	✓	✓

Table 5. shows how the solutions respond on the given attacks. N/A – the attack is irrelevant; Dash – the attack is relevant but the solution does not provide the countermeasure; Check Mark – the attack is relevant and the solution provides the countermeasure against it.

success is negligible in security parameter. To demonstrate the soundness of their schemes authors use different approaches. The Solution 1 - PoW uses simluation-based approach, and claims that the scheme is secure with soundness $\left(\frac{L-d+1}{L}\right)^t$ where L is reduce buffer, t is number of challenges on Merkle-Tree and d is the minimum distance of the linear code. The variables $L, d\ and\ t$, all are tunable and leads to the negligible probability of adversary's success to convince the verifier. The Solution 2 - s-PoW, also uses simulation-based approach and demonstrates the negligible probability of adversary to cheat the verifier. $P(succ) = (1 - \varepsilon(1 - g))^K$, where ε is a fraction of the file that is unknown for the attacker and g is the probability to guess unknown bit correctly and K is the length of response bit string. It is possibility to choose K (depends on security parameter), in such a way that the $P(succ)$ became negligible in security parameter. Security analyses in Soltuion 3 - POF is superficial. The scheme provides the proof that the cheating of the scheme is at least as hard as "performing strong collision attack of the hash function". And the probability of collision attack is calculated based on Birthday Paradox[31]. The probability of collision $P = 1 - \epsilon^{\frac{-k(k-1)}{2N}}$, where k is number of inputs, N is a number of all possible hash values(outputs) and $\epsilon = 2.71828$ is an euler's number .(e.g for SHA-256 probability of collision $p = \epsilon^{\frac{-k(k-1)}{2*2^{256}}} \approx \epsilon^{-\frac{k^2}{2*2^{256}}} \approx \epsilon^{-\frac{1}{2}(\frac{k}{2^{128}})^2}$). The security analyses of the Solution

---

[31] "Given k randomly generated values, where each value is a non-negative integer less than N, what is the probability that at least two of them are equal?" `http://preshing.com/20110504/hash-collision-probabilities/` last seen October 3, 2016

4 - CSD, differs from other solutions, as it uses the game-based security proof approach. Based on two players A - adversary and A_* – extractor, is defined, that the scheme is secure in $(\varepsilon_0, \varepsilon_1)$ if the probability of guessing b single bit by extractor A_* plus some negligible in security parameter $negl(\lambda)$, is grater or equal then probability of adversary A guessing the same b bit: $Pr[b_A = b] \leq Pr[b_{A_*} = b] + negl(\lambda)$.Where ε_0 is a minimum min-entropy of the file and $(\varepsilon_0 - \varepsilon_1)$ – is a max length of bits adversary is allowed to learn. The security analyses are practically missing (or are provided very briefly) in Solutions 5 and do not cover the probability of success of adversary. The last two solutions both offer the security analyses and demonstrate the probability of adversary convincing the verifier, which are negligible in security parameter and depend in both cases on J – number of tokens generated by prover as the response on the challenge received from verifier. For Solution 6 - bf-POW the probability of success is $(p + (1-p)(0.5^l) + p_f(1 - 0.5^l))^J$. While for Solution 7 - ce-POW $P(succ) = (p + (1-p) * 0.5^l)^J$. Where in both cases p is the probability that adversary knows the randomly chosen chunk, g is the probability to guess the randomly chosen byte, l is a token size and p_f in Solution 6 is a probability of false positive in BF.

Besides the soundness of the scheme, it is significant to underline the leakage resilience and restrictions of the schemes. The most of the solutions defines a threshold T of data leakage, and if the adversary learns up to T bits, the security of the scheme still holds(e.g. PoW the threshold is set to $64Mb$). In some solutions the adversary can communicate with accomplices and receive the information up to given threshold, but it should take place before or after the protocol is running. While the last solution (ce-POW) allows receiving the information about file only before the protocol starts. It is important to underline that in PoW, the files are take from specific class of distribution, while later schemes build their security definition based, on arbitrary file distribution. The summary of security analyses, is provided in the table below.

---

[32] L is reduce buffer, t is number of challenges on Merkle-Tree and d is the minimum distance of the linear code.

[33] ε is a fraction of the file that is unknown for attacker and g is the probability to guess unknown bit correctly and K is the length of response bit string.

[34] .ε_0 is a minimum min-entropy of the file and $(\varepsilon_0 - \varepsilon_1)$ – is a max length of bits adversary is allowed to learn

[35] p is the probability that adversary knows the randomly chosen chunk, g is the probability to guess the randomly chosen byte, l is a token size and p_f is a probability of false positive in BF

[36] p is the probability that adversary knows the randomly chosen chunk, g is the probability to guess the randomly chosen byte, l is a token size

	Input File	Leakage Resilience	Soundness
Solution 1 PoW	Specific Class of Distribution	✓	$\left(\frac{L-d+1}{L}\right)^t$ [32]
Solution 2 s-PoW	Arbitrary Distribution	✓	$P(succ) = (1-\varepsilon(1-g))^K$[33]
Solution 3 POF	N/A	–	Probability of Collision in Hash Function
Solution 4 CSD	Arbitrary Distribution	✓	$Pr[b_A = b] \le Pr[b_{A_*} = b] + negl(\lambda)$[34]
Solution 5 OpenStack based	Arbitrary Distribution	N/A	Security Based on Merkle-Tree lemma[20]
Solution 6 bf-POW	Arbitrary Distribution	✓	$P(succ) = (p + (1-p)(0.5^l) + p_f(1-0.5^l))^J$[35]
Solution 7 ce-POW	Arbitrary Distribution	✓	$P(succ) = (p + (1-p) * 0.5^l)^J$[36]

Table 6. Comparison: Security Features

The results demonstrate that most preferable choices in terms of security are the solution 4 and the solution 7. They ensure that the client and the server both are the honest players. Both avoid the file and content detection attacks by implementing the proof of ownership, avoid confidential data profiling by implementing encryption and avoid poisoning attack by implementing additional check over encrypted files. Their security is build on the files taken from arbitrary distribution and allow the predefined amount of data leakage, which prevents from CDN attack. On the other hand less preferable choices are solution 3 and Solution 5. The Solution 3 does not cover the honest-but-curios-server's issue and also the authors do not provide the discussion about leakage resilience in a straightforward manner. There for it is also not clear to measure the tolerance to the CDN attack. As regards to the Solution 5 - the paper covers the security analyses of the scheme superficially and do not provide the probability of success of adversary. It is also hard to calculate the leakage resilience of the scheme. But the solution addresses the data privacy issues in the remote storage and uses encryption and Merkle-Tree to avoid the cheating from both players(client and server). The drawback of Solution 2 and Solution 6 is that they consider the server as a honest player and does not ensure the client data privacy. But both of them have the clear image of the probability of success of adversary to cheat the scheme and it is calculated based on arbitrary distributed files in both cases. A disadvantage of the first solution-PoW is that it builds the security on the assumption that the files are taken from the specific distribution class and it is hard to prove such assumption. Also it does not address the honest-but-cures-server's issues.

5.2. Cost Analyses

The aim of the data deduplication is the reduction of bandwidth and disk space consumption. All the solutions covered above, carry the extra calculation in order to maintain the security. It increases the bandwidth and storage consumption in comparison with not secure "hash-as-a-proxy" solution. But those seven solutions still are more efficient then "whole file transfer" approach. This section evaluates the additional generated data traffic and disk space for each scheme, and calculates the cost based on the Amazon S3 Pricing.[37]

In the Solution 1 – PoW, the server asks for the randomly chosen 20 leaves and their sibling paths of the Merkle-Tree. Sibling path consists of the sibling node of the given leaf and also all the parent and their sibling node values, from the sibling leaf till the roof of the tree. So the client generates the sibling path $P = (v_s, v_{n_0}, v_{n_1}, ..., v_{n_i})$, where v_s is a value of the sibling node [38], i is the hight of the Merkle-Tree and $v_{n_0}, v_{n_1}, ..., v_{n_i}$ are the values of the parent nodes. The maximum buffer size is $64MB$. The leaf size is 256 bits. So in worst case the Merkle-Tree has $\frac{64*1024*1024*8}{256} = \frac{2^6*2^{10}*2^{10}*2^3}{2^8} = 2^{21}$ leafs and the height of tree is 21.The size of sibling path is $2*21*256 = 10752$ bits $\approx 1.3KB$. The traffic generated by PoW is 20 times sibling path size, which is approximately $26KB$ "Data Transfer In" operation for remote storage. As regards to the extra disk space consumption, the server requires to store the root value (plus number of leaves, which is negligible size) – output of hash function, $256\ bits$ long string (hash function used in implementation is SHA256).

For the Solution 2 – s-POW – generated traffic includes: the random seed s, which server sends to the client and the client's response: K bits long string called $response$. s is an integer $0 < s < file\_size$, so it is negligible(4 bytes), and the response length is tunable, the maximum size used in paper is 1830 bits. The traffic generated during the proof is approximately $0.23KB$. To look at the space consumption, for each file

---

[37] `https://aws.amazon.com/s3/pricing/`, last seen October 3, 2016

[38] The value of the node of the Merkle-Tree is the corresponding block if the node is a leaf or the hash of the left and right children if it is an intermediate node.

it is a tuple the server keeps in the database. The tuple contains 4 elements: ptr – the pointer on the file –8 byte long. $res[]$ – an 10000 length array of generated challenges, called "responses" – each for 1830 bits length string; id_c – the highest challenge computed so far – integer 2 bytes; id_u – number of challenges used so far– integer 2 bytes and finally it uses the hash of the file as a key. The following formula calculates total extra space: 256 bits (hash of file) + 64 bits(ptr) + $10000 * 1830$ bits(array) + 32 bits (id_c) + 32 bits (id_u) = 18300384 bits $\approx 2.29MB$.

During the POF protocol in the Solution 3, the client and the server exchange the following data: The hash value of file $h(F)$ (hash function is SHA256) ; R_c the random number $R_c \leftarrow_R \{0,1\}^*$ – for calculation consider the length of the R_c to be 256 bits; Two random seeds – $S_1 \leftarrow_R \{0,1\}^*$ and $S_2 \leftarrow_R \{0,1\}^*$ – consider them as 256 bits long; The integer c – number of blocks – 32 bits long; Client confirmation of obtaining the session key $h_{K_s}(R_c, TS)||TS$ – concatenation of hash value and TimeStamp – output of SHA256 plus the value of unix Timetamp(32 bit length); And finally the summary value, the proof– hash of concatenation of hashes of chosen blocks – 256 bits long (again using the SHA256). To sum up the generated data, we will have a following result: total data transfer In = 256 bits (hash of file) + 256 bits (hash of R_c with session key) + 32 bits(TimeStamp)+ 256 bits (the proof) = $100\ Bytes$ and total data transfer Out = 32 bits (c number of blocks) + 256 bits (R_c) + $2 * 256$ bits (S_1 and S_2) = $80\ Bytes$. The solution 3 generates in total approximately $0.1KB$ traffic, which is very efficient. And the protocol is even more efficient in terms of the disk space consumption, it occupies only extra $256\ bits$ (hash value of file) per file same as "hash-as-a-proxy" solution.

The Solution 4 adds encryption over the proof of ownership protocol and generates the following traffic. The client sends the hash of the file $hash(F)$ to the server - 256 bits string. The server challenges the client and sends back the encrypted AES key C_τ– less then 256 bits length string. To prove the ownership of the file the client sends the hash value of AES encrypted file – also the size is 256 bits. The generated traffic per file is totaling to: $2 * 256 = 512\ bits = 64\ Bytes$ for data transfer In and $32\ Bytes$ - data transfer Out. Disk space required for protocol consists of the encrypted file and the meta-data. The size of encrypted file is approximately same as the plain file.[39] And the size of meta-data is the size of the following key-

---

[39] The AES-256 output calculation: $CipherText_{size} = PlainText_{size} + Block_{size(256)} - (PlainText_{size}\ MOD\ Block_{size(256)})$

value pair: $(key = hash(F); value = (hash(C_F), C_\tau))$. Total 256 bits ($hash(F)$) + 256 bits($hash(C_F)$) + 256 bits($C_\tau$) $\approx 0.1KB$. The additional storage is $\approx 0.1KB$.

Data exchange between the client and the server in the Solution 5 looks as following: the client first sends the Merkle-Tree root value over the encrypted file to the server – 256 bits string. The server request the super-logarithmic number of leaves and the sibling paths, same as the Solution 1 scenario , meaning the approximately $26KB$ data transfer In. And if the proof holds then the server sends the URI of the requested file. Swift restriction – length of container is maximum $256Bytes$ and the maximum length of object name is $1024Bytes$.[40] Hence the generated data transfer Out is $1280\ Bytes$. As for extra disc space – it is only $256\ bits$ long and represents the Merkle-Tree root value of encrypted file.

In Solution 6 the bandwidth and storage consumption, both are depending on token size. If token size is small – bandwidth consumption is low, while the required disk space grows. The growth of tokens size cause the growth of data traffic, but the decrease of storage consumption. In worst case scenario the extra storage consumption is $2MB = 2000KB$ for the tokens 128 bits length, but it could be decreased to $40\ bytes = 0.04KB$. In order to prove the ownership the following data is generated: 256 bits (the hash of the file) + $J * 32$ bits (the J length array($pos[]$) of the integers)+ $J * 1024$ bits (the J length array($res[]$) of $tokens$(max length of token size in paper is 1024 bits) = 256 bits + $1040 * J$ bits. Where $J \in \{102,\ 204,\ 509,\ 1017\}$. Taking into consideration a worst case scenario, the generated data during the bf-POW protocol is $256 + 1024 * 1017 = 1041664\ bits \approx 130KB$ In and $32 * 1024 = 32768\ bits \approx 4KB$ Out data transfer, while it could be decreased till $256 + 128 * 102 = 13312\ bits \approx 1.7KB$ In and $32 * 128 = 512\ Bytes$ Out data transfer.

The last solution Solution 7 is similar to the Solution 6 in terms of bandwidth consumption. First client sends the file identifier – output of SHA-1 160 bits. Then server challenges the client and sends the J length $pos[]$ array of integers – $J * 32$ bits. In response the client sends back to the server the J length array $res[]$ of $tokens$(max length of token size in paper is 1024 bits). And the $J \in \{91,\ 182,\ 457,\ 914\}$. In worst case scenario the ce-POW protocol generates $160+914*32 \approx 3.6KB$ data transfer Out and $1024 * 914 + 160 = 936096 \approx 117KB$ data transfer In. The additional disk space

---

[40] http://docs.openstack.org/developer/swift/api/object_api_v1_overview.html, last seen October 3, 2016

required for protocol is the sum of the identifier (output of SHA1 – 160 bits), 10000 length array of J numbers($10000 * 914$ bits) and same length array of pre-calculated responses ($10000 * 914 * 1024$ bits). Total = $160 + 10000 * 914(1 + 1024) = 9368500160$ bits ≈ $1.1GB$.

In order to demonstrate the extra cost of the solutions, the table contains the calculation using the Amazons S3 - simple monthly calculator [41]. To have an approximate picture of volume of the average remote storage providers, I have refereed to the statistics of one of the most popular storage provider – Dropbox.[42]. The numbers are impressive, Dropbox daily receives 1.2 billion files and by the year of 2015, it was storing 35 billion Microsoft Office files. Taking into consideration also the deduplication ration – $4 : 1$[43] (this is an average ratio, when the nature of data is unknown) we can assume that the remote storage provider could duplicate daily $\frac{1200000000}{\frac{3}{4}} = 900000000$ files – 30 monthly, and store more the 35 Billion files.(As 35 billion is the number of only Microsoft office files and the user's content is much more reach then just the office files)

In terms of the cost we can see from the table that most efficient is Solution 1 - PoW, which has the same expense as the non secure "Hash-as-a-proxy" solution. After comes Solution 3, with \$ 39.33 in month. Third and fourth places are respectively for Solution 4 and Solution 5. The Solution 6 is on fifth position with \$ 2335.65 per month and after come the Solution 2 and Solution 7 where the price dramatically grows and is practically not affordable.

5.3. Summary

The section Evaluation compared the security characteristics and additional costs for required bandwidth and storage of the solutions. We have identified the most preferable solution from the security point of view and they are: Solution 4 and Solution 7. As the Solution 7 is no affordable and the cost of the Solution 4 is quite realistic – \$ 107.34 per month, we can see that the Solution 4 is the optimal choice for

---

[41] `https://calculator.s3.amazonaws.com/index.html` last seen September 29, 2016

[42] `http://expandedramblings.com/index.php/dropbox-statistics/`, last seen September 29, 2016

[43] `https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Tivoli+Storage+Manager/page/Deduplication+FAQ` last seen September 29, 2016

	Traffic	Storage	Total Cost
Solution 1	$26KB * 9*10^8 = 23.4TB(In)$	$256\ Bits * 35*10^9 = 1.12TB$	$ 34.2 *see the monthly bill*
Solution 2	$0.23KB * 9*10^8 = 193GB(In)$	$2.29MB * 35*10^9 = 80.15PB$	$ 2390882.94 *see details*
Solution 3	$100\ Bytes * 9*10^8 = 90GB(In)$ $80\ Bytes * 9*10^8 = 72GB(Out)$	$256\ Bits * 35*10^9 = 1.12TB$	$ 39.33 *see the monthly bill*
Solution 4	$64\ Bytes * 9*10^8 = 57.6GB(In)$ $32\ Bytes * 9*10^8 = 28.8GB(Out)$	$0.1KB * 35*10^9 = 3.5TB$	$ 107.34 *see the monthly bill*
Solution 5	$26KB * 9*10^8 = 23.4TB(In)$ $1280\ Bytes * 9*10^8 = 1.15TB(Out)$	$256\ Bits * 35*10^9 = 1.12TB$	$ 128.84 *see the monthly bill*
Solution 6	$130KB * 9*10^8 = 117TB(In)$ $4KB * 9*10^8 = 3.6TB(Out)$ or $1.7KB * 9*10^8 = 1.5TB(In)$ $512Bytes * 9*10^8 = 430GB(Out)$	$1.7KB * 35*10^9 = 59.5TB$ or $132KB * 35*10^9 = 4.62PB$	$ 2335.65 *see details* or $ 145198.87 *see details*
Solution 7	$117KB * 9*10^8 = 105.3TB(IN)$ $3.6KB * 9*10^8 = 3.24TB(Out)$	$1.1GB * 35*10^9 = 38500PB$	$ 1143495895.96 *see details*
"Hash-as-a-proxy"	$32\ Bytes * 9*10^8 = 28.8GB(In)$	$256\ Bits * 35*10^9 = 1.12TB$	$ 34.2 *see the monthly bill*

Table 7. Data traffic and storage costs based on Amazon S3 pricing. Estimated amount of files and generated traffic is offered according the static of one of the most popular remote storage provider - Dropbox.

implementation. But if the Hones-but-curios servers is not an issue then it is better to choose the Solution 1 as it has the lowest cost. Despite the low cost still not preferable choices are the Solution 3 and the Solution 5 because of their security characteristics. Choosing the Solution 6 is also not reasonable, because it has the same security features as the first solutions, in difference that the input files are taken from arbitrary distribution, but the cost is 68 times more. In most cases the decision which solution to choose depends on implementation's objectives, but in any case solutions 2, 6 and 7 are out of the game, because of extremely high cost.

In real life scenario to take a decision, it is also important to have the results of I/O and CPU efficiency. To obtain the result, it is required to implement all the solution in same environment, which is out of scope of this work.

6. Conclusions

This study is dedicated to the privacy and confidentiality issues in remote storage, that uses the Data Deduplication technology. The trivial solution is to refuse to use the Data Deduplication at all. But this technology has a lot of benefits in terms of cost reduction, and the cloud service providers are keen to look some alternative solutions. The main requirement for those solutions is to hold desired security level, while not to increasing the cost of provided services significantly. The authors of papers addressing this issue revealed that the problem in Data Deduplication is that the process is detectable, and to prove the data ownership a small piece of information – hash of the content – is used. This thesis gathers the solutions, which are proposing to substitute the hash as a proxy approach with interactive protocols, in order to prove the data ownership in remote storage.

The thesis covers seven solutions, and provides the detail overview of their functionality. It takes into consideration that the cloud service providers are looking for cost efficient solutions, but at the same time it is important to hold the security. In this work we provide security and cost analyses. The security analyses identifies the security characteristics of the solutions and compares them to each other based on the following criteria:

- How resilient is the solution to the attacks listed in the thesis?
- How resilient is the solution to the data leakage?
- What is the soundness of the protocol? What is the probability that adversary passes the protocol?
- Does the soundness holds for arbitrary distribution of file or it works for the specific class of distribution?

The study declares the Solution 4 and the Solutions 7 as the most preferable choices, from security point of view.

The cost analyses involves the bandwidth and the disk space cost calculation for each solution. After diving into the operational part of the solutions, it was easy to calculate the required bandwidth and disk space for reach solution. We used the statistics from Dropbox to have the idea about the scale of the remote storage (amount of files stored on the disk) and we use the Amazon Web Services' price list to see the cost of each solution. To see the result of the analyses refer to the Table 7. To take into consideration the result of both analyses the study suggests to implement the Solution 4, as a secure and affordable solution.

The decision – which solution to implement in the remote storage, depends on the priority of the remote storage itself. In some cases the key factor is price or performance efficiency, while in other confidentiality and privacy plays an important roll. This study helps to make the decision according to the cost and security criteria, while CPU and I/O efficiency is out of scope of this work.

References

[1] B. P. Shai Halevi, Danny Harnik and A. Shulman-Peleg, “Proofs of ownership in remote storage systems,” *ACM Conference on Computer and Communications Security*, pp. 491–500, 2011.

[2] A. S. Roberto Di Pietro, “Boosting efficiency and security in proof of ownership for deduplication,” *the 7th ACM Symposium on Information, Computer and Communications Security*, pp. 81–82, 2012.

[3] J. Z. Jia X, Ee-Chien Changn, “Weak leakage-resilient client-side deduplication of encrypted data in cloud storage,” *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 195–206, 2013.

[4] M. L. Nesrine Kaaniche, “A secure client side deduplication scheme in cloud storage environments,” *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–7, 2014.

[5] A. O. Lorena Gonzalez-Manzano, “An efficient confidentiality-preserving proof of ownership for deduplication,” *Journal of Network and Computer Applications*, vol. 50, pp. 49–59, 2015.

[6] A. O. A. S. Jorge Blasco, Roberto Di Pietro, “A tunable proof of ownership scheme for deduplication using bloom filters,” *Communications and Network Security (CNS), 2014 IEEE Conference on*, pp. 481–489, 2014.

[7] J. M. Chao Yang, Jian Ren, “Provable ownership of file in de-duplication cloud storage,” *2013 IEEE Global Communications Conference (GLOBECOM)*, pp. 695–700, 2013.

[8] M. Dutch, “Understanding data deduplication ratios,” pp. 5–9, 2009.

[9] M. B. Babette Haeusser, Alessio Bagnaresi and A. Woodcock, *Guide to Data Deduplication:The IBM System Storage TS7650G ProtecTIER De-duplication Gateway*. IBM Redbooks, 2008, pp. 20–27.

[10] S. Srinivasan, *Security, Trust, and Regulatory Aspects of Cloud Computing in Business Environments*. USA: Information Science Reference (an imprint of IGI Globalo), 2014, pp. 77–78.

[11] M. H. T. K. M. R. U. V. Moritz Borgmann, Tobias Hahn and S. Vowe, *On the Security of Cloud Storage Services*. Germany: SIT Technical Reports, 2012, pp. 55–110.

[12] B. P. Danny Harnik and A. Shulman-Peleg, "Side channels in cloud services, the case of deduplication in cloud storage," *IEEE Security and Privacy Magazine*, vol. 8, no. 2, pp. 40–47, 2010.

[13] L. Reyzin, "Some notions of entropy for cryptography," *Information Theoretic Security*, vol. 6673, pp. 138–142, 2011.

[14] R. C. Merkle, "A certified digital signature," *In Proceedings on Advances in cryptology*, vol. CRYPTO, no. 89, pp. 218–238, 1989.

[15] P. B. Miltersen, "Universal hashing," *Lecture note*, p. 12, 1998.

[16] W. Dai, "Crypto++ library, 5.6.1," last accessed - Jan 2011. [Online]. Available: http://www.cryptopp.com/

[17] D. L. E. M. Mark Storer, Kevin Greenan, "Secure data deduplication," *ACM international workshop on Storage security and survivabilit*, pp. 1–10, 2008.

[18] NIST, "Announcing the advanced encryption standard (aes)," 2001.

[19] J. P. J. W. Can Wang, Zhi-guang Qin, "A novel encryption scheme for data deduplication system." *Communications, Circuits and Systems (ICCCAS), 2010 International Conference on*, pp. 265–269, 2010.

[20] R. C. Merkle, "A digital signature based on a conventional encryption function." *Advances in Cryptology — CRYPTO '87*, vol. 293, pp. 369–378, 1988.

[21] A. M. Darrel Hankerson and S. Vanstone, *Guide to Elliptic Curve Cryptography.* Springer-Verlag New York, 2003.

[22] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors." *Communications of the ACM*, vol. 13, pp. 422–426, 1970.

A. Appendix 1

POF – PERFORMANCE MEASUREMENTS AND COMPARISON

	PoW (ms)			POF (ms)		
Size (MB)	Disk Read	Merkle Tree	Total	Disk Read	Algorithm	Total
0.015625	0.09	0.57	0.66	0.15	0.62	0.77
0.03125	0.13	1.07	1.2	0.16	0.42	0.58
0.0625	0.19	1.73	1.92	0.17	0.62	0.79
0.125	0.34	4.01	4.35	0.2	0.63	0.83
0.25	0.62	6.82	7.44	0.24	0.63	0.87
0.5	1.17	12.51	13.68	0.27	0.58	0.85
1	2.03	21.08	23.11	0.29	0.62	0.91
2	4.44	42.46	46.9	0.31	0.62	0.93
4	8.19	84.46	92.65	0.55	0.63	1.18
8	14.76	168.43	183.19	0.66	0.65	1.31
16	28.62	334.88	363.5	0.82	0.64	1.46
32	56.75	669.38	726.13	1.32	0.67	1.99
64	112.58	1352.01	1464.59	4.34	0.64	4.98
128	223.08	2692.07	2915.15	5.56	0.65	6.21
256	437.84	5393.64	5831.48	2.11	0.65	2.76
512	1269.46	10932.49	12201.95	5.53	0.64	6.17
1024	2581.56	23344.83	25926.39	5.52	0.63	6.15

Table 8. Client Computation Time – POF vs PoW [7]