# Trust-Oriented Protocol for Continuous Monitoring of Stored Files in Cloud

Alexandre Pinheiro[1], Edna Dias Canedo[2], Rafael Timóteo de Sousa Júnior[1] and Robson de Oliveira Albuquerque
[1]Electrical Engineering Department
[2]Faculdade UnB Gama
University of Brasília, UnB
Brasília - DF, Brazil
E-mail: tenpinheiro@dct.eb.mil.br, ednacanedo@unb.br, desousa@unb.br, robson@redes.unb.br

*Abstract* — **The speed, availability, scalability and low cost are main attractive of cloud services. However, building safe storage services from a customer point of view, mainly when this service is being hosted on public cloud infrastructure, whose service providers are not fully trustworthy, it is an obstacle to be overcame. There are common situations, where owners of large data amount need to store them for a long time, but they will not necessarily need to access them. This time can vary from few years to decades, in accordance with applicable laws of each country. In these cases, important aspects as integrity, availability and privacy must be considered when making decision on adoption of cloud services. Considering the damage whose information loss or its leakage may cause, this paper presents a protocol, which through an independent checker, allows a periodic monitoring on stored files in cloud using trust and cryptography concepts to ensure data integrity. Moreover, this paper also presents a protocol reference implementation and the performed tests results.**

***Keywords-protocol; trust; cloud data storage; integrity; data monitoring.***

## I. INTRODUCTION

Companies, institutions and government agencies generate large amounts of information in digital format, such as documents, projects, transactions records etc., every day. For legal or business reasons, this information needs to remain stored for a long period of time and this has become an issue for IT managers.

The use of cloud services for storing sensitive information started to gain relevance, along with its popularization, cost reductions and an ever-growing supply of Cloud Storage Services (CSS). However, ensuring integrity and confidentiality still has to be evaluated in such services in order to protect information.

CSS for data storage are fast, cheap, and almost infinitely scalable. However, reliability can be an issue, as even the best services sometimes fail [1].

A considerable number of organizations consider security and privacy as obstacles to the acceptance of public cloud services [2].

Data integrity is defined as the accuracy and consistency of stored data. This condition indicates that the data has not changed and has not been broken [2]. CSS should provide mechanisms to confirm data integrity, while still ensuring user privacy.

Considering these perspectives, this paper proposes a protocol based in outsourced service which provides the CSS customers the constant assurance of the existence and integrity of their files without the need to keep copies of the original files or expose its contents.

This paper is structured as follows: Section II reviews works related to data integrity in the cloud. Then, Section III proposes a new protocol named Trust-Oriented Protocol for Continuous Monitoring of Stored Files in Cloud (TOPMCloud). A detailed analysis of the TOPMCloud is shown in Section IV. The Section V shows a TOPMCloud implementation. A resume of obtained results are presented in Section VI. Section VII ends this paper with some conclusions and outlines future works.

## II. RELATED WORK

In order to try to guarantee the integrity of data stored in CSS, many research works suggested solutions with both advantages and disadvantages regarding the domain analysed in this paper.

The protocol proposed by Juels and Kaliski [3] enables the CSS to prove a file subjected to verification was not corrupted. To that end a formal and secure definition of proof of retrievability was presented and introduced the use of sentinels. Sentinels are special blocks, hidden in the original file prior being encrypted and then used to challenge the CSS. In the work of Kumar and Saxena [4], a scheme was presented, based on [3] where one does not need to encrypt all the data, but only a few bits per data block.

George and Sabitha [5] proposed a solution to improve privacy and integrity. The solution was designed to be used in tables and it was divided in two parts. The first, called ‘anonymisation’ is used to identify fields in records that could identify their owners. Anonymisation uses techniques such as generalisation, suppression, obfuscation, and addition of anonymous records to enhance data privacy.

The second, called ‘integrity checking’, uses public and private key encryption techniques to generate a tag for each record on a table. Both parts are executed helped by trusted third party called ‘enclave’ that saves all the data generated that will be used for deanonymisation and integrity verification.

A new encrypted integrity verification method is proposed by Kavuri et al. [6]. The proposed method uses a new hash algorithm, the Dynamic User Policy Based Hash Algorithm. Hashes on data are calculated for each authorised cloud user. For data encryption, an Improved Attribute-Based Encryption algorithm is used. Encrypted data and its hash value are saved separately in CSS. Data integrity can be verified only by an authorized user and it is necessary to retrieve all the encrypted data and its hash.

Another proposal to simultaneously achieve data integrity verification and privacy preserving is found in the work of Al-Jaberi and Zainal [7]. Their work proposed the use of two encryption algorithms for every data upload or download transaction.

The Advanced Encryption Standard (AES) algorithm is used to encrypt client's' data which will be saved in a CSS, and a RSA-based partial homomorphic encryption technique is used to encrypt AES encryption keys that will be saved in a third party together with a hash of the file. Data integrity is verified only when a client downloads one's file.

In the work of Kay et al. [8], a data integrity auditing protocol that allows the fast identification of corrupted data using of homomorphic cipher-text verification and recoverable coding methodology was proposed. Due the methodology adopted, both the total auditing time and the communication cost could be reduced. Checking the integrity of outsourced data is done periodically by a trusted or untrusted entity.

In the work of Wang et al. [9], it is presented a security model for public verification for storage correctness assurance that supports dynamic data operation. The model guarantees that no challenged file blocks should be retrieved by the verifier during the verification process and no state information should be stored at the verifier side between audits. A Merkle Hash Tree (MHT) is used to save the hashes of authentic data values and both the values and positions of data blocks are authenticated by the verifier.

In the work of Jordão et al. [10], an approach was presented that allows inserting large volumes of encrypted data in non-relational databases hosted in the cloud and after that performs queries on inserted data without the use of a decryption key. Although not the main focus of the work, this approach could be used to verify the integrity of stored content in the cloud through the evaluation of responses to queries with previously calculated results.

The proposed solutions in [5][7][8][9] are using asymmetric cryptographic algorithms which are admittedly slow compared to symmetric algorithms. The solution proposed by Kavuri et al. [6] needs to retrieve whole file to check it. In the work of Juels and Kaliski [3], and in the work of Kumar and Saxena [4], small changes in the file can remain unnoticed until the whole file to be recovered.

Thus, unlike the works cited above, this paper presents a protocol that only uses symmetric encryption to check file integrity. Furthermore, it uses a challenge/response-based technique for checking the integrity without download the file. Finally, our solution checks all file bytes so that any change, no matter how small, will be identified quickly.

## III. PROTOCOL OBJECTIVES

The main objective of protocol is to make possible utilization of an outsourced service allowing client to constantly monitor the integrity of their stored files in CSS, without having to keep copies from original files or reveal its contents.

### A. *Protocol requirements*

One of the main requirements of this protocol is to prevent the CSS provider from offering and charging a client for a storage service that in practice is not being provided. Other premises are low bandwidth consumption, quick identification of a misbehaving service, providing strong defenses against fraud, avoiding the overloading of CSS, ensuring data confidentiality and also giving utmost predictability to the Integrity Check Service (ICS).

### B. *Protocol operating principle*

The basic operating principle of the protocol begins with the encryption of the original file, followed by its division into 4096 small chunks, which in turn are grouped randomly to form each data block with distinct 16-chunks. Hashes will be generated from these data blocks and together with the addresses of the chunks which formed the data block are sent to the ICS.

The selection and distribution of chunks used to assemble the data blocks is done in cycles. The number of cycles will vary according to the file storage period. Each cycle generates 256 data blocks without repeating chunks.

The data blocks generated in each cycle contains all of the chunks of the encrypted file (256 * 16 = 4096).

Each hash and its chunk addresses will be used only once by the ICS to send an integrity verification challenge to the CSS provider. On receiving a challenge with the chunk addresses, the CSS reads the chunks from stored file, assembles the data block, generates a hash from data block and sends the hash as answer to the ICS.

Chunks number per file as well as chunks number to compose each data block was chosen through mathematical simulations. These simulations seek to find small numbers that minimize the required time to fully check a file, but large enough to make it impossible to save hashes from all possible data blocks without taking up more disk space than the original file.

To finalize, the answer hash and the origin hash are compared by ICS. If the hashes are equal, it means the content of evaluated chunks in the stored file is intact.

### C. *Protocol architecture*

The protocol architecture is based in three components: i) customers; ii) storage services in the cloud; and iii) an integrity check service. The interaction between the architectural members is carried out through an exchange of asynchronous messages.

The protocol consists of two distinct processes. The first, called "File Storage Process", which is run on demand and has the client as its starting point. The second called 'Verification Process' is instantiated by an ICS and executed continuously to verify one CSS.

An ICS can simultaneously verify more than one CSS through parallel instances of the Verification Process. An overview of the TOPMCloud protocol architecture is shown in Figure 1.

#### 1) *File Storage Process*

File Storage Process is responsible for preparing the file to be sent to the CSS and for generating the information needed for its verification by ICS.

In Figure 1, each stage from 'File Storage Process' are named with the prefix 'Stage FS-' followed by the number of the stage and, if necessary, by its stage sub-process number.

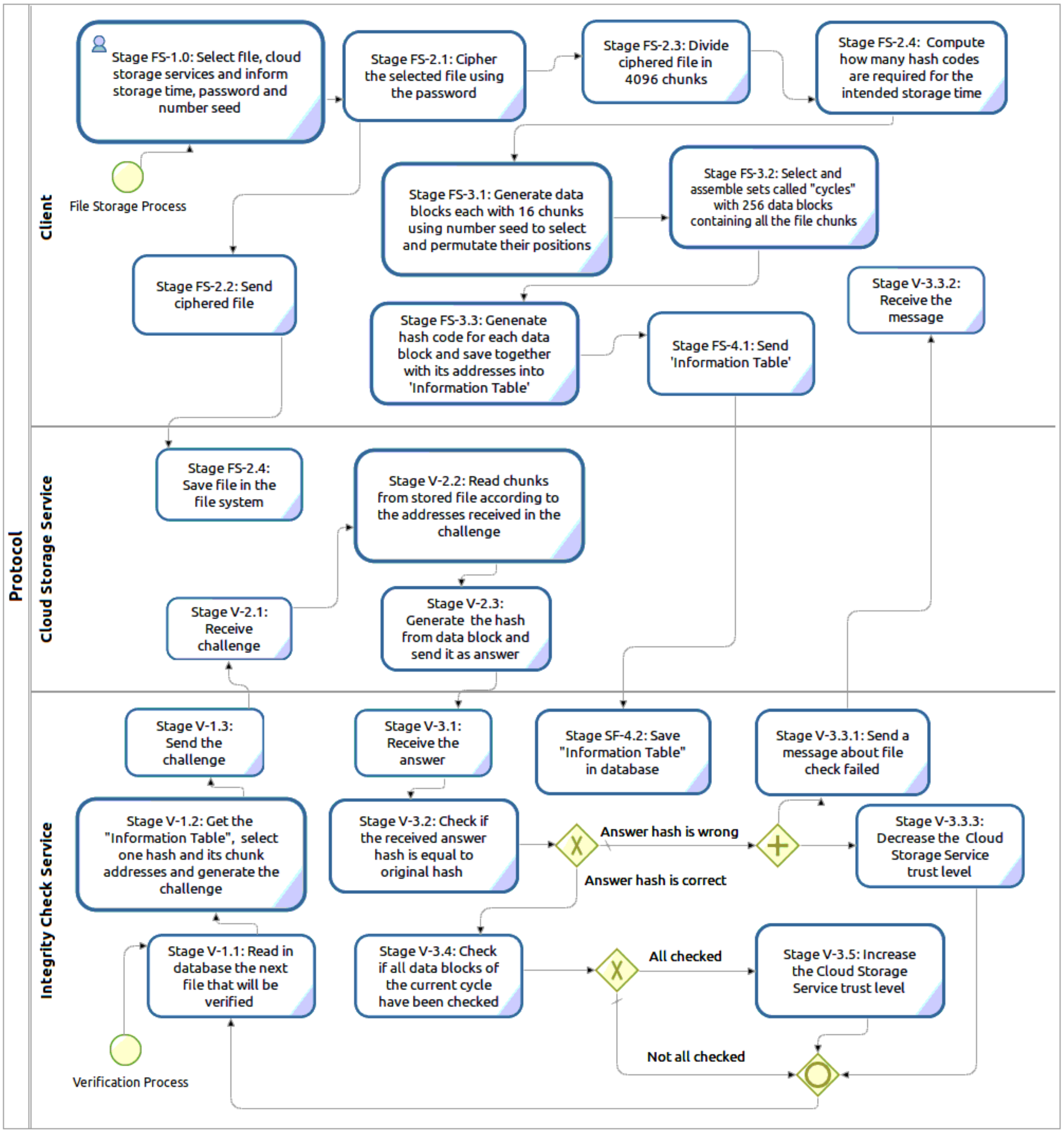
Protocol
Client
Cloud Storage Service
Integrity Check Service
Stage FS-1.0: Select file, cloud storage services and inform storage time, password and number seed
File Storage Process
Stage FS-2.1: Cipher the selected file using the password
Stage FS-2.3: Divide ciphered file in 4096 chunks
Stage FS-2.4: Compute how many hash codes are required for the intended storage time
Stage FS-2.2: Send ciphered file
Stage FS-3.1: Generate data blocks each with 16 chunks using number seed to select and permutate their positions
Stage FS-3.2: Select and assemble sets called "cycles" with 256 data blocks containing all the file chunks
Stage FS-3.3: Generate hash code for each data block and save together with its addresses into 'Information Table'
Stage FS-4.1: Send 'Information Table'
Stage V-3.3.2: Receive the message
Stage FS-2.4: Save file in the file system
Stage V-2.2: Read chunks from stored file according to the addresses received in the challenge
Stage V-2.1: Receive challenge
Stage V-2.3: Generate the hash from data block and send it as answer
Stage V-1.3: Send the challenge
Stage V-3.1: Receive the answer
Stage SF-4.2: Save "Information Table" in database
Stage V-3.3.1: Send a message about file check failed
Stage V-1.2: Get the "Information Table", select one hash and its chunk addresses and generate the challenge
Stage V-3.2: Check if the received answer hash is equal to original hash
Answer hash is wrong
Answer hash is correct
Stage V-3.3.3: Decrease the Cloud Storage Service trust level
Stage V-1.1: Read in database the next file that will be verified
Stage V-3.4: Check if all data blocks of the current cycle have been checked
All checked
Not all checked
Stage V-3.5: Increase the Cloud Storage Service trust level
Verification Process

Fig. 1. TOPMCloud Protocol.

The process goes as follows. First, the user should select a file, a password, a number seed, and the time in years for which it intends to maintain the monitoring of file integrity. The password will be used to generate the secret key used to encrypt chosen file.

The number seed will add extra entropy to the process that creates a random seed used to warrant an unpredictable selection and distribution of the data that forms the source of the hash codes to be used to check file integrity.

One or more CSS should also be selected. Considering the need to ensure the recoverability of files, selecting more than one provider is important to provide redundancy, given that customers will not keep a copy of the original files.

In the next stage, data blocks are generated from random union of 16 chunks of the original file. For this, after the split, each file chunk receives an address code between 0 and 4095, represented in hexadecimal format (000 - FFF).

An algorithm developed for this purpose will make the selection of chunks and permutation of their positions for each data block.

The algorithm uses the number seed provided by user to creates the random seed which will be used to add an extra layer of the entropy to the pseudo-random sequence generator that chooses the sequences of address codes.

The algorithm is also responsible for grouping data blocks in sets called ‘cycles’. Each cycle consists of 256 data blocks that have all the 4096 chunks of the original file. This stage is shown in Figure 1 as sub-processes ‘FS-3.1’, ‘FS-3.2’, and ‘FS-3.3’.

Finally, in the last stage is built a data structure named 'information table’. It contains record header made up by a file identifier, the chunk size in bytes, the body records amount, the total number of cycles, and a checksum. Each record on the table represents a data block and contains fields with the address codes of the 16 file chunks, their cycle number and hash code. The ‘information table’ is sent to ICS and the file sent to CSS is deleted from the customer.

*2) Verification Process*

This process is designed to periodically check the integrity of files saved in CSS. Furthermore, it assigns a trust level for each CSS according to the check results.

Trust is recognized as an important aspect for decision-making in distributed and auto-organized applications [11][12]. Marsh [11] provided a clarification of trust concepts, presented an implementable formalism for trust, and applied a trust model to a distributed artificial intelligence (DAI) system in order to enable agents to make trust-based decisions.

Trust and security have become crucial to guarantee the healthy development of cloud platforms, providing solutions for concerns such as the lack of integrity and privacy, the guarantee of security and author rights.

The verification process consists of the following stages: In ICS, selecting the next file to be checked, generating the challenge and delivering it to CSS; In CSS, receiving the challenge, reading the chunks from saved file according to the challenge, assembling the data block, generating the data block hash, rendering and sending the answer to ICS; and in ICS again, receiving and checking the answer. The sub-process from “Verification Process” shown in Figure 1 follows the same previously used rules, but are prefixed with ‘Stage V-’.

In the first stage, the ICS verifies what next file should have its integrity checked in a given CSS, performing the same procedure in parallel with each other registered CSS.

After selecting the file, its information table is read and the number of the last checked cycle is retrieved. When the file is new or when the last checked cycle has already been completed, a new and not yet checked cycle is randomly chosen.

After that, the next not-verified record that belongs to the selected cycle is selected from information table. The challenge is assembled using the address codes obtained from selected record, the file chunk length, the file identifier and a challenge identifier. When ready, the challenge is sent to CSS and the pool of challenges that are waiting for an answer is updated.

In the second stage, the CSS receives the challenge and concurrently retrieves all chunks from saved file. Chunks are retrieved according to address codes and length received in the challenge. All chunks retrieved are concatenated forming a data block, and from it, a hash is generated. This hash is packaged together with the challenge identifier and sent as response to the ICS.

In the third stage, the ICS receives the answer, finds the challenge in the pool, reads the original record from the information table and compares the received hash with the hash that gave rise to the challenge. If they do not match, a message is sent to client reporting the error. Whenever a file verification process fails, the CSS trust level is immediately downgraded. The sub-processes from ‘V-3.1’ to ‘V-3.6’ show this stage in Figure 1.

When the ICS does not receive an answer from the CSS on a challenge, after the wait time has expired, the original challenge is re-sent and the wait time is squared. After the 10th unsuccessful attempt, the challenge is considered failed and the same procedures described in the third stage are adopted.

If the response hash and the original hash are equal, then a flag will be saved in the information table record, indicating that the data block represented by its hash was successfully verified. After that, case there is no other record in the current cycle to be checked, this means that all of data blocks of the file saved in CSS have already been successfully verified and the CSS trust level must be raised.

To end the stage, the trust level classification process will be done. Upon completion of this stage, the process is re-executed from the first stage.

*3) Trust Level Classification Process*

Whenever a verification process fails, the trust level of the CSS verified will be downgraded. When the current trust level value is greater than zero, it is set to zero, when the trust value is in the range between 0 and -0.5, it is reduced by 15%. Otherwise, it is calculated the value of 2.5% from the difference between the current trust level value and -1, and the result is subtracted from trust level value. These calculations are shown in the source code below.

```
IF  (TrustLevel > 0) THEN
     TrustLevel = 0
ELSE IF (TrustLevel >= -0.5) THEN
     TrustLevel = TrustLevel - (TrustLevel * -0.15)
ELSE
     TrustLevel = TrustLevel - {[(-1) - TrustLevel] * -0.025}
```

However, whenever a checking cycle is completed without failures (all the data blocks of a file have been checked without errors), the CSS trust level is raised. If the current trust level value is less than 0.5, then the trust level value is raised by 2.5%. Otherwise, it is calculated the value of 0.5% from the difference between 1 and the current trust level value, and the result is added to trust level value. These calculations are shown in the source code below.

```
IF (TrustLevel < 0.5) THEN
    TrustLevel = TrustLevel + (TrustLevel * 0.025)
ELSE
    TrustLevel = TrustLevel + {[1 - TrustLevel] * 0.005}
```

The trust level in a CSS will affect the verification periodicity of its files and the time needed to get a complete file cycle. When the verification of all the data blocks in a cycle has been successfully concluded, this means that all the chunks of the original file were tested.

The minimum percentages of stored files on each service that will be verified per day, as well as the minimum percentage of data blocks that will be checked by file and per day, according to trust level, are shown in Table I. The values 1 and -1 that respectively represent blind trust and complete distrust are incompatible with the object of classification and will not be considered.

TABLE I. CLASSIFICATION OF THE TRUST LEVELS

| **Trust Level** | **Value Range** | **Files verified by day** | **% from file verified by day** | **Data Blocks verified by day** |
|---|---|---|---|---|
| Very high trust | ]0.9, 1[ | 15% | ~ 0.4% | 1 |
| High trust | ]0.75, 0.9] | 16% | ~ 0.8% | 2 |
| High medium trust | ]0.5, 0.75] | 17% | ~ 1.2% | 3 |
| Low medium trust | ]0.25, 0.5] | 18% | ~ 1.6% | 4 |
| Low trust | ]0, 0.25] | 19% | ~ 2.0% | 5 |
| Low distrust | ]-0.25, 0] | 20% | ~2.4% | 6 |
| Low medium distrust | ]−0.5, −0.25] | 25% | ~ 3.2% | 8 |
| High medium distrust | ]−0.75, −0.5] | 30% | ~ 4.0% | 10 |
| High distrust | ]−0.9, −0.75] | 35% | ~ 4.8% | 12 |
| Very high distrust | ]-1, -0.9] | 50% | ~5.6% | 14 |

Whenever the value of the trust is zero, a fixed value is assigned to determine the initial trust. Thus, if the last check resulted in a 'positive assessment', a value of +0.1 is assigned for trust; otherwise, if a fault has been identified, the assigned value is -0.1.

## IV. PROTOCOL ANALYSIS

As a prerequisite to define the characteristics of the proposed protocol we took into consideration the following assumptions: low consumption of network bandwidth; predictability and economy in consumption of ICS resources; fast identification of misbehaving services; privacy; resistance against fraud, and no overloading of the CSS.

Thus, the proposed logical division of the file into 4096 chunks, grouped into blocks of 16 chunks each, aims at minimizing the storage service overhead by reducing the amount of data to be read for each verification, and enabling the parallel execution of searching and recovering each data chunk.

Fast identification of badly behaved services also helped to determine the proposed values. The protocol uses a random selection of 16 file chunks in the data block, to allow checking the integrity of various parts of the file in a single verification step.

Privacy is attained with the use of 256-bit hash codes to represent each data block, regardless of their original size. The hash codes allow the ICS to perform the validation of files hosted in storage services, without necessarily knowing their contents.

Furthermore, the use of hash codes in combination with a fixed amount of data blocks, providing predictability and low usage of the network bandwidth. It is possible to pre-determine the computational cost required to verify the integrity of a file, the whole time foreseen for its storage, regardless of its size.

There is also the possibility to predict the total number of data blocks, as it varies according to the time predicted for the file storage, so that each hash code and the chunk addresses that formed the data are used only once, uniquely and exclusively as a challenge to the CSS. Calculations were made based on a worst-case scenario, i.e., the hypothetical situation where the CSS remains throughout the file storage period rated as 'Very High Distrust'.

According to Table 1, in a CSS rated as 'Very High Distrust', it is necessary to check at least 14 data blocks of each file a day. As the data blocks generation is performed in cycles with 256 blocks each, to determine the total number of data blocks to be generated (2), it is necessary to first calculate the total number of cycles (1).

$$Cycles = ROUND\left(\frac{(14*366)*Years}{256},\ 0\right) \tag{1}$$

$$Data\ Blocks\ =\ Cycles\ *\ 256 \tag{2}$$

From the definition of the number of blocks, it is possible to determine the size of the 'information table' and, therefore, the computing cost to transfer and store it in an ICS.

Finally, fraud resistance is obtained by means of a selection and swapping algorithm that assigns the entropy needed to render as impracticable any attempt to predict which chunks are in each data block, as well as the order in which these chunks were joined. A brute force attack, generating hash codes for all possible combinations of data blocks, is not feasible as the number of possible combinations for the arrangement of 4096 file chunks in blocks with 16 chunks each is of about $6.09 \times 10^{57}$ blocks (3). Consequently, to generate and store 256-bit hash codes for all possible combinations of data blocks would need about $1.77 \times 10^{47}$ TB in disk space (4).

$$A_{n,p} = \frac{n!}{(n-p)!} \rightarrow A_{4096,16} = \frac{4096!}{(4096-16)!} \cong 6.09 \text{ x } 10^{57} \tag{3}$$

$$Disk\ Space\ (TB) = \frac{(6.09 \text{ x } 10^{57}) * 256}{8*1024^4} \cong 1.77 \text{ x } 10^{47} \tag{4}$$

## V. IMPLEMENTATION

The implementation of TOPMCloud protocol was divided in three stages. At first stage, all processes of customer responsibility were implemented. At second stage, all processes of ICS responsibility were implemented, and finally, at third stage, all processes of CSS responsibility were implemented.

For each stage, we developed an application, all of them using components of Java EE technology as JPA, EJB and CDI. For the client, we developed a desktop application, whereas for ICS and CSS, we developed Web Service (WS) applications. The utilized application server was Glassfish and, as Database Management System (DBMS), we chose PostgreSQL.

The main customer application tasks are: encrypting the file; dividing it into equal chunks; assembling data blocks; generating their hashes and joining them in cycles; generating the information table and sending it to the ICS; and, finally, sending the encrypted file to CSS. For task of encrypting file, we chose the AES-256 algorithm using Cipher-Block Chaining (CBC) operation mode.

Raffle process of file chunks to compose each data block was implemented using the SHA1PRNG algorithm, a pseudo-random number generator. Due to its high speed, security, and simplicity, we adopted the algorithm Blake2 [13] as cryptographic hash function for this work.

At second stage, we implemented all necessary routines for ICS provider executing the file monitoring using WSs. The main tasks implemented by ICS application are: receiving information tables and saving them in your database; generating challenges and sending them to CSSs; monitoring and receiving answers about challenges; and ranking trust level from monitored CSSs.

At third and last stage, as well as in previous stage, we implemented all necessary routines to the CSS answering the challenges to ICS using WSs. The main tasks in CSS application are: receiving and saving challenges in your database; processing challenges; and sending response to ICS. Challenges reception, their processing and responses delivery are carried out in parallel and asynchronously.

## VI. RESULTS

The scenario used to test the application and to verify TOPMCloud efficiency was composed of three Virtual Machines (VM) (a customer, a ICS and a CSS). These VMs shared resources from a computer equipped with an Intel core i7 2.4 GHz, 12GB of RAM memory, and a 1TB SATA Hard Disk 5400 RPM.

Five tests were performed with three different large file sizes. The average results are presented in Table II.

TABLE II. TEST RESULTS

| File size | Times spent to: | | | |
|---|---|---|---|---|
| | **encrypt file** | **compute file hash** | **compute data block hashes (for 1 year)** | **check one data block** |
| 5 GB | 4.0 min | 23.4 s | 32.0 min | 0.11 s |
| 9 GB | 7.1 min | 91.1 s | 71.8 min | 0.88 s |
| 14 GB | 10.6 min | 137.0 s | 108.6 min | 1.20 s |

In order to simulate a file integrity breakdown and determine the number of days required to identify the fault, tests were performed on a CSS classified as 'low trust'. The same three previously tested files were monitored, however only the 5 GB file had changed its contents. The characteristics and results of each test are presented in Table III.

TABLE III. FAULT SIMULATION TESTS

| Bytes changed | File percentage | Location | Affected chunks | Days required to find fault | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | **T1** | **T2** | **T3** | **T4** | **T5** | **Average** |
| 4688 | 0.000085% | random choice | 1 or 2 | 10 | 9 | 22 | 12 | 15 | ~14 |
| 55006658 | 1.0% | end of file | 41 | 3 | 7 | 6 | 6 | 3 | 5 |

With the aim to determine the maximum number of days required to complete a file check cycle (all its data blocks being checked at least once) on each trust level, we did simulations using the percentages defined for each trust level in Table I. The results are presented in Figure 2.

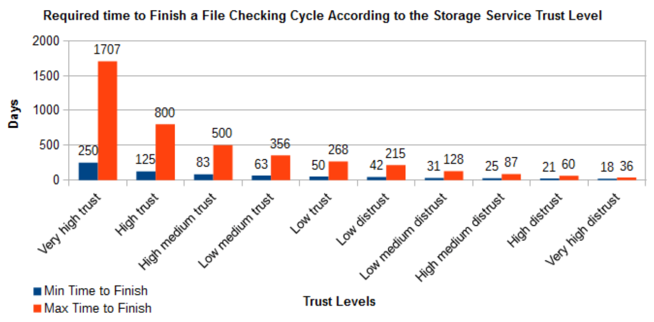

Fig. 2. Number of days required to complete a file check cycle.

This time variation is intended to reward services with fewer failures, minimizing consumption of resources such as processing and bandwidth. Moreover, it allows prioritizing the protocol for checking files that are stored in services that have already failed, reducing the time required to determine if there are any corrupted or lost files.

## VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a protocol to ensure the integrity of stored files in CSS through an ICS hosted by a third party. The protocol named TOPMCloud allows periodic and qualified monitoring of file integrity using trust concepts without confidentiality compromising.

Based on CSS behaviour, the checking frequency can increase or decrease, reducing overload on services that never fail or checking more quickly all stored files in CSSs that have already failed.

As it can be seen in Section VI, TOPMCloud provides a efficient control over file integrity. Even on very large files, time spent checking each data block does not overload the CSS. Variation in number of days required to identify the same fault, which was purposely inserted in the file, obtained in subsequent tests, confirms the randomness of the selection process of each data block sent as a challenge to CSS.

Another important result obtained was the speed to identify a fault, even interfering in only 2 from 4096 file chunks, the average period for its identification was only two weeks.

As part of future works, we intend to formalize and validate the protocol using Petri nets. We will conduct tests simulating the protocol behaviour with CSSs classified in all trust levels. Furthermore, we want to add a mechanism to ensure that, only when authorized by a customer, a ICS could send challenges to CSS.

## REFERENCES

[1] S. T. Tandel, V. K. Shah, and S. Hiranwal, "An implementation of effective XML based dynamic data integrity audit service in cloud," in International Journal of Societal Applications of Computer Science, vol. 2, issue 8, pp. 449-553, 2014.

[2] P. Dabas and D. Wadhwa, "A recapitulation of data auditing approaches for cloud data," in International Journal of Computer Applications Technology and Research (IJCATR), vol. 3, issue 6, pp. 329-332, 2014.

[3] A. Juels and B. S. Kaliski, "Pors: proofs of retrievability for large files" in 14th ACM Conference on Computer and Comunication Security (CCS), Alexandria, VA, pp.584-59, 2007.

[4] R. S. Kumar and A. Saxena, "Data integrity proofs in cloud storage," in Third International Conference on Communication Systems and Networks (COMSNETS), Bangalore, pp. 138-146, 2011.

[5] R. S. George and S. Sabitha, "Data anonymization and integrity checking in cloud computing," in Fourth International Conference on Computing (ICCCNT), Communications and Networking Technologies, Tiruchengode, pp. 758-769, 2013.

[6] S. K. S. V. A. Kavuri, G. R. Kancherla, and B. R. Bobba, "Data authentication and integrity verification techniques for trusted/untrusted cloud servers," in International Conference on Advances in Computing, Communications and Informatics (ICACCI), New Delhi, pp. 2590-2596, 2014.

[7] M. F. Al-Jaberi and A. Zainal, "Data integrity and privacy model in cloud computing," in International Symposium on Biometrics and Security Technologies (ISBAST), Kuala Lumpur, pp. 280-284, 2014.

[8] H. Kay, H. Chuanhe, W. Jinhai, Z. Hao, W. Xi, L. Yilong, Z. Lianzhen, and W. Bin, "An efficient public batch auditing protocol for data security in multi-cloud storage," in 8th ChinaGrid Annual Conference (ChinaGrid), Changchun, pp. 51-56, 2013.

[9] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "An enabling public verifiability and data dynamics for storage security in cloud computing," in 14th European Symposium on Research in Computer Security (ESORICS), Saint-Malo, pp. 355-370, 2009.

[10] R. Jordão, V. A. Martins, F. Buiati, R. T. Sousa Jr, and F. E. Deus, "Secure Data Storage in Distributed Cloud Environments," in IEEE International Conference on Big Data (IEEE BigData), Washington DC, pp. 6-12, 2014.

[11] S. P. Marsh, “Formalising Trust as a Computational Concept”, Ph.D. Thesis, University of Stirling, 1994.

[12] T. Beth, M. Borcherding, and B. Klein, “Valuation of trust in open networks,” in 3Th European Symposium on Research in Computer Security (ESORICS), Brighton, pp. 1-18, 1994.

[13] J. Aumasson, S. Neves, W. O’Hearn, and Z. Winnerlein, "BLAKE2: Simpler, Smaller, Fast as MD5," in Applied Cryptography and Network Security, Springer Berlin Heidelberg, pp. 119-135, 2013.