A Reliable Secure Storage Cloud and Data Migration Based on Erasure Code

Emmy[1] Mugisha and Gongxuan Zhang[2]
[1]School of Computer Science and Engineering, Nanjing University of Science and Technology
Naning, 210094 - China
[e-mail: emymugi@gmail.com]
[2] School of Computer Science and Engineering, Nanjing University of Science and Technology
Naning, 210094 - China
[e-mail: gongxuan@njust.edu.cn]
*Corresponding author: Emmy MUGISHA

Received May 4, 2017; revised June 17, 2017; revised August 29, 2017; accepted September 15, 2017; published January 31, 2018

Abstract

Storage cloud scheme, pushing data to the storage cloud poses much attention regarding data confidentiality. With encryption concept, data accessibility is limited because of encrypted data. To secure storage system with high access power is complicated due to dispersed storage environment. In this paper, we propose a hardware-based security scheme such that a secure dispersed storage system using erasure code is articulated. We designed a hardware-based security scheme with data encoding operations and migration capabilities. Using TPM (Trusted Platform Module), the data integrity and security is evaluated and achieved.

Keywords: Storage cloud, erasure code, TPM, information storage and retrieval, data security

1. Introduction

Fast Internet generation and Internet of things (IoT) usability posed public attention in recent decades. Several services are derived from the Internet domain so that users can gain access with on-demand concept. The concept of cloud computing is a resource pool through Internet. Cloud tenants apparently gain access to storage cloud services without good knowledge on control and technical aspects of the whole cloud system. In this work, we look at the design of a storage cloud system that is rooted in hardware-based security for data confidentiality and integrity. A storage cloud system is composed of independent storage nodes dispersed in different remote geographical locations. Data security and integrity are emerging concerns for storage cloud design. Several researchers proposed their ideas in pushing data file to storage nodes [1, 2, 3, 4, and 5].

Replication technique is considered as a mean to provide data integrity. This is achieved by replicating a file (data) into a random number of replicas in a way that each storage node will accommodate one file replica. However, for replication concept, given that a storage node is down and is no longer accessed, the replica that it accommodates will never be restored anymore. Therefore, applying erasure code (EC) as a file distribution technique can improve data integrity. This is assured by encoding a file f of k symbols into a codeword of n symbols. Moreover, EC will then distribute individual codeword symbols to distinct storage nodes on the storage cloud architecture. Given that a node is compromised, its codeword symbol is directly equal to an erasure error. With erasure code computation, it can recover compromised storage nodes when the fixed number of possible EC recoverable nodes is reached. The file can be recovered from the codeword symbols dispersed in non-compromised storage nodes under EC decoding process. EC computes the file into codeword symbols independently. However, it is best applicable in a dispersed storage system. In the encoding process, when a data symbol (file) is pushed to the storage node, the node independently derives a codeword symbol corresponding to the receipted data symbol and stores it. Also, the recovery process is the same as that of the encoding.

Pushing data to the storage cloud poses much attention regarding data confidentiality. To implement robust data confidentiality over a storage node, each node must employ trusted platform module (TPM) functions with which a log file will be stored. Given that a tenant initiates an access request to his file stored in the storage cloud node, the TPM functions employed on this storage node must authenticate the request source by comparing it with the pre-measured parameters (log file) stored in the destination storage node's TPM. When authentication process succeeds, the tenant will retrieve the codeword symbols from the requested storage nodes, and the EC decode process follows. Using TPM functions will avoid several storage security problems. Firstly, for encryption data security is easy to be compromised due to its software-based security scheme, key sharing and protection is managed by tenants through Internet medium. Secondly, long-term storage systems based on encryption are replaced decades by decades, which end up by failing re-encryption processes due to new updated strong cryptographic algorithms.

2. Background

2.1 Trusted Platform Model

In this paper, we introduce a hardware-based security scheme using Trusted Platform Module (TPM). TPM is a standardized dedicated microcontroller explored to secure hardware by integrating cryptographic keys into devices. Moreover, the TPM technical specification was implemented by Trusted Computing Group (TCG). After that, in 2009 the specification was standardized as ISO/IEC 11889 by the International Organization for Standardization (ISO) and International Electrotechnical Commission [6]. Furthermore, TCG revised the TPM version 1.2 specifications and on the 13 March 2014 it was published [7]. However, the proposal revision for the TPM version 2.0 specifications was also published on March 13, 2014, for public review. Hence, the latest TPM 2.0 specification revision was released in October 2014 [8].

TPM provides functions for the secure generation of cryptographic keys and limits their use in addition to a random number generator [9][10]. Moreover, it also includes capabilities such as remote attestation, binding, and sealed storage. The remote attestation generates a hash key summary of the hardware and software configurations which allows a third party to verify that the software has not been changed. Next, binding encrypts data using a unique RSA key (TPM binding key) from a storage key [11]. Lastly, Sealing is also TPM capability (TPM_Seal) that encrypts data in the same way as binding does but the difference is that it forces TPM to be in a certain state before the data is decrypted (TPM_Unseal) [12]. Therefore, cloud applications can use TPM to authenticate, secure hardware devices and data respectively since each TPM chip has a unique and secret RSA key burned into it during manufacture. Hence it is capable of a platform or device authentication as well as secures data storage.

2.2 TPM Application

In cloud computing scenario, putting the security down to the hardware level together with software based solutions contributes improved protection than applying only a software-based solution [13].

In addition, the United States Department of Defense (DoD) set a limit that its new computer assets manufactured to support DoD must employ a TPM version 1.2 or above. Consequently, the TPM is capable of device measurement, integrity, identification, encryption and authentication [14]. The following are selected but not limited areas of TPM application or use.

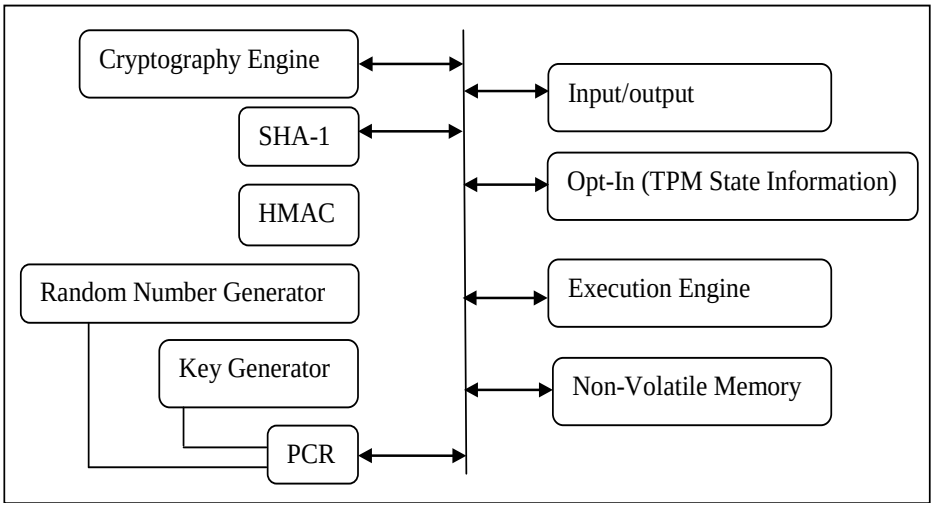

Fig. 1. TPM simplified architecture

a) Device or platform integrity

The primary concept of a TPM is to assure the integrity of a platform. That is to say, in its concept the term integrity implies that a platform or device must behave as intended to. However, the platform is considered as any computer platform and not limited to PCs or a particular operating system. Moreover, TPM compose platform configuration registers (PCR) that allow secure storage and reporting of security measurements. In addition to that, these measurements are used to monitor any change of pre-measured configurations that determines the next platform's activity. For instance, in Linux Unified Key Setup (LUKS) [15] and Microsoft's BitLocker Drive Encryption and PrivateCore vCage memory encryption is described.

Therefore, the cloud provider's BIOS and the operating system should take the primary responsibility to utilize TPM in order to assure their cloud architecture integrity. From here, applications and users running on that cloud can rely on its security features, i.e. secure I/O, uncompromised keyboard entries, memory and storage operations.

b) Disk encryption

In full disk encryption applications, i.e. BitLocker Drive Encryption, dm-crypt and SecureDoc can employ this TPM technology to store and protect the keys used to encrypt the computer hard disks that provide integrity authentication for a trusted boot pathway. For instance BIOS, boot sector, and etc.

c) Password protection

In a general normal access to keys and data, the authentication process is always required by entering a password phrase. And, if the authentication process is implemented at a software level only, the access typically is prone to dictionary attacks. However, since

TPM is implemented in a hardware module, a dictionary attack prevention mechanism was presented. And this effectively protects against automated dictionary attacks and still allow the user to try as many as the sufficient and reasonable number of tries. Therefore, with this hardware based dictionary attack prevention, the user can choose shorter or weaker password phrases which are easier to remember. That is to say, without this hardware security level of protection, only password phrases with longer and high complexity would provide sufficient protection.

In this position paper, we target the problem of data integrity and security by pushing authenticity and security measures to the hardware level. We consider security control on a system that is composed of distributed storage nodes. Relying on encryption algorithms over storage systems through Internet channels is a risky concept. As a solution to distributed systems, we define TPM functions per storage node independently. Specifically, these nodes are required to perform all storage cloud service authentications. Moreover, with TPM functions, we address a new security scheme based on hardware levels together with erasure code to build a robust, secure storage cloud system. The erasure code concept implements file (data) encoding steps and migration from one tenant/node to another. The definition of encoding, TPM, and data migration on this cloud achitecture will result a relaible storage system with data integrity and confidentiality. Therefore, we look at the system in a wide setting that permits easy modification between the number of storage nodes and stable.

Given a cloud with n dispersed storage nodes, a file is halved into *k* data blocks and symbolized as a variable of *k* symbols. To this far, the ideal is to construct a secure storage cloud environment that afford secure data migration using hardware-based security through node-to-node TPM authenticity scheme. The TPM authenticity scheme with erasure codes to support file distribution will produce a robust and relaible storage cloud architecture that assures data migration between tenants. The dispersed storage nodes will independently perform encoding process and data migration under the umbrella of TPM node-to-node authenticity. Here, the wide setting of a secure storage cloud environment introduced. We set $n = \alpha k$ to approximate the number of storage nodes to be higher than number of data blocks (*k*). That is to say, the number of storage nodes *n* is bigger than *k*. Hence, the more the number of data blocks, the more data availability and EC efficiency.

2.3 Erasure Code

Erasure coding (EC) is a method for protecting data (a file). Data is broken into fragments (blocks), it then expanded and encoded into redundant data pieces and stored across a set of different locations or storage media **[**16**]**. However, the goal of erasure coding is to enable data that becomes corrupted at some point in the storage process**,** to be reconstructed/recovered by using information about the data that is stored in other nodes of the same cloud. Moreover, erasure codes are often used instead of traditional RAID **[17]** because of their ability to reduce the time and overhead required to reconstruct data in the storage cloud architecture. Its drawback is that it can be more CPU-intensive, and that can translate into increased latency. Nevertheless, erasure coding can be useful with large

quantities of data and any applications or systems that need to tolerate failures, such as disk array systems, data grids, distributed storage applications, object stores and archival storage. In addition, the commonly current use case for erasure coding is object-based storage cloud.

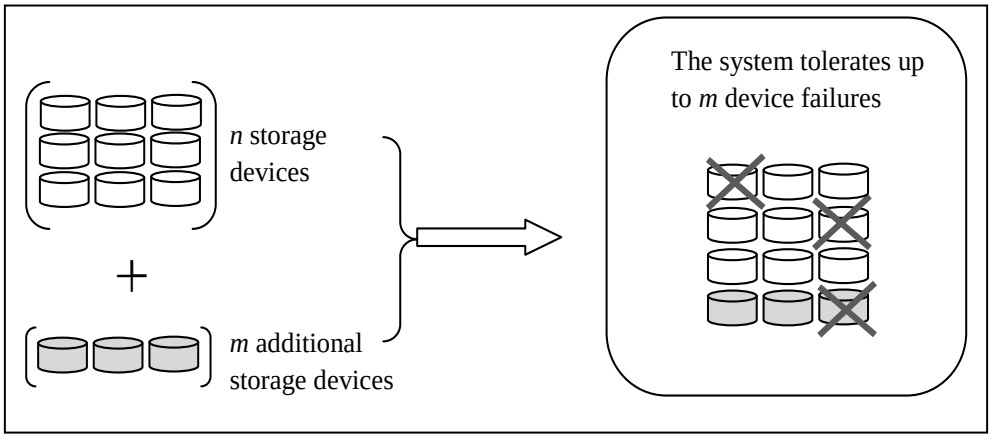

Fig. 2. Erasure code overview

Erasure coding creates a mathematical function to describe a set of numbers so that they can be checked for accuracy and recovered if one is lost. This function is referred to polynomial interpolation or oversampling as the key concept behind erasure codes. Going deeper, in mathematical terms the protection offered by erasure coding can be represented in simple form by the following expression: $n = k + m$, where variables k is the original amount of data or symbols and m is the extra or redundant symbols that are added to provide prevention from system failure. Furthermore, the variable n is the total number of symbols created after the erasure coding process. For example, in a 10 of 16 configurations, or EC 10/16, six extra symbols (m) should be added to the 10 base symbols (k). Consequently, the 16 data fragments (n) will then be spread across 16 drives, nodes or geographic locations. Hence, the original file could be reconstructed from 10 verified fragments.

Much more, erasure codes *is* also known as forwarding error correction (FEC) codes, it has been developed more than 50 years ago. However, different types have emerged since that time and in one of the earliest and most common types is Reed-Solomon [18]. From this type, the data (a file) can be reconstructed using any combination of k symbols even if m symbols are unavailable. Based on the above example, in EC 10/16, six drives, nodes or geographic locations might be lost or unavailable, and the original file would still be recoverable.

3. Related Work

3.1 Dispersed Storage Systems

Considering decades back, the Network-Attached Storage (NAS) [19] with the Network File System (NFS) [20] provided spare storage machines via network targeting tenants that would access these machines using network connectivity. Moreover, some advanced research findings on security, robustness, availability, efficiency, and scalability were presented in [1, 2, 21 and 22]. A non-centralized storage system is scalable due to individual storage node that behaves on and off without a central control authority. As a result, about storage node failures, an easy solution is to replicate a file and push the replicas into distinct storage nodes. Also, this solution is expensive based on the increase of a certain number of replicas in a specific number of iterations with time. To solve this issue, erasure code is introduced to encode a file into codeword symbols [5, 19, 23, 24, and 12]. A file is encoded as a variable of symbols, whereby individual storage node keeps a codeword symbol. Therefore, a storage node failure is an equivalent of erasure error of the pushed file codeword symbol. Random linear codes allow dispersed encoding and each codeword symbol is severally derived.

Storing a file of k data blocks, a storage node randomly assigns a chosen coefficient to these data blocks, and then pushes both the codeword symbols and coefficients. To retrieve the file, a tenant initiates a request to k storage nodes to retrieve the pushed file codeword symbols and coefficients, hence compute the linear systems. Dismakis et.al [25] looks at an event where $n = ak$ and a is immutable value. They depicted out that dispersing a block of a file to q randomly selected storage nodes is sufficient with a probability $k/p - o(1)$ of a restored file, where $q = b \ln k > 5a$, and p is the prime order of the used group. Therefore, setting $q = b \ln k$ is equal to n storage nodes to which a file data blocks are sent to. As q increases, the connectivity cost between nodes and file codeword transfer becomes expensive as well as retrieval probability the better. Going deeper, the design show weak data confidentiality and this opens a door for an adversary to compromise k storage nodes, hence getting the exact data (m). Lin and Tzeng [26] designed strong data confidentiality problems by implementing a secure decentralized erasure code for the storage systems. Moreover, their architecture is composed of dispersed storage servers and key servers for cryptographic key management. In this architecture, stored files are encrypted and then encoded under erasure code settings. Whenever there is a need for a file, the user requests storage servers through the key servers. The user gain access to the stored file when n available key servers are over a threshold t with best probability.

In **[27]**, a Secure Multi-Agent based framework for Communication among Open Clouds is presented. In this framework, the secure interoperability, portability, and the communication bebween different number of clouds is introduced within cloud computing concept. They urgue that increasing the scope of clouds in-term of processing, infrastructure, application, utility and other services, inter-communication within clouds is

needed. And for single cloud architecture implementation, they considered it impossible in-terms of commutation, and that security comes first, and considered the most important element to enhence the communication. Therefore, to resolve this issue they proposed the Multi based Framework for Secure and Reliable Communication among Open Clouds. That, all the MAs (Mobile Agents) are registered with directory services, and every MAs of the cloud interceipts the information about each other after being registered in the directory.

Comparing their solution with our work (on a single cloud), their MAs is our Nodes and DA is our main node (**Fig. 5**) with which all nodes are registred and communicate to each through the main node under TPM authentication mechanism (**Fig. 3**) per node on the network. They urgue that the Directory Agent provides advantegeous linkages between MAs and security in addition compared to traditional frameworks, in our opinion all frameworks that still implement software based security solutions are regarded vulnerable to adversaries. Therefore, in this case our scheme is designed in the way that each node implements TPM capabilities and its TPM provenance information or identification is pre-measured at the initail stage and submited to the main node's TPM, therefore our authentication occures at the TPM level which is not the case for them. Their DA is based on software security level which makes their IDs and other identification matrics vulnerable, while our main node is based on harware security level and no adversary can penetrate it.

The BDMSC design integrates computing-networking technological platforms, which allow fully adaptive energy-efficient joint reconfigurations of the virtual resources available at data centers and mobile devices under hard real-time constraints, to cope with unpredictable volume of data generated by this emerging applications was introduced in **[28]**. In their work, they identified both the major opportunities and challenges of the BDSMC paradigm. In a practical view, they presented the StreamCloud architecture and prototype to contextualize their simulation results. The reported results concluded that the resource management framework implemented by StreamCloud in the real-world might attain noticeable energy savings. Compared to our model, the BDMSC design show high potential use of distributed computing resources, and its performance and energy-efficiency play a big role in the world of cloud computing. However, the orchestration of BDMSC architecture does not detail data flow security mechanism and computing resources integrity per inter-networked resource to establish trust between inter-communicating networked-computing resources.

We urge that applying TPM-Based security over BDMSC architecture may strengthen trust and integrity perspectives.

Similarly, the FoE (Fog-of-Everything) by **[29]** looks at the spectrum of hierarchically-organized networked computing nodes (Fog and remote Cloud data centers) to cope with the increasingly large volume of data from IoE-based applications. The FoE also highlite adaptive energy-efficient reconfiguration and orchestration of the virtualized computing-plus-communication resources available at the computing nodes and thing

devices under real-time constraints on the allowed computing-plus-communication delay and service latency.

To archieve this, they introduced FoE model. This model consider that in the next years, IoE devices will be equipped with multiple (possibly, heterogeneous) wireless network interface cards. And, that this will open the doors to the design of energy-efficient transport protocols, that rely on the emerging Multipath TCP paradigm [30]. Therefore, the increment of the per-connection throughput, while limiting the energy overhead induced by the parallel utilization of multiple radio interfaces, should be considered. Henceforth, that the native selforganizing feature of the IoE model induces hierarchical relationships among the involved things **[31]**. As a result, the design of new Network-layer communication primitives for IoE-based ecosystems is required in order to implement suitable forms of selective multicast that account for the relative roles of the involved IoE devices [31]. Unfortunately, FoE lack innovative solutions tackling distributed security, trustworthyness and thing authentication that is needed in order to allow the migration of the FoE paradigm from the theory to the practice level. This position paper addresses hardware-based security to the provisioning of integrity among cloud architectural networked resources which on the other hand would be applied to FoE model as a result of strengthening trust in addition to latency and energy- efficiency.

3.2 Integrity Computation Concept

Data and service integrity checking is a kennly looked at concept in the world of cloud computing, in particular storage cloud systems. Moreover, when a tenant pushes his file data to the storage cloud, there is no more way to control it given that the tenant desires to check the status of his data, i.e. if the data is still the same that was stored initially to the storage nodes. The concept of data integrity checking [32 and 33] and the opinion of cogent evidence of storage trends [34, 35, and 36] are presented. However, public stored data auditing was addressed in Wang et al. [37].

4. System Model and Design

Our system model consists of tenants, control node (main node that perform file distribution) and *n* dispersed storage nodes $N_i, N_{i+1}.....,N_n$. Storage nodes allow storage services and authentication processes for data security and integrity provisioning. We show our system output in four categories as follows: system apparatus, data storage, data migration, and data restoration. First, describing system apparatus category, the system controller selects system settings to which the whole system will base on to serve tenants and other system functionalities.

A system node implements TPM functions. Each node is assigned a TPM identity (stored in the log file) which differentiate it from other nodes on the same network (**Fig. 5**). Second, data storage category, a tenant prepares his file *f* and sends it to main storage node. A file is then segmented into *k* blocks $B_i, B_{i+1},...., B_{k-1}$ where $i = 0$ and *f* has a unique source identity

(ID). After segmentation process, the main node pushes each block to *q* randomly chosen storage nodes (**Fig. 5**). Moreover, when a block is receipted at a storage node, each storage node individually performs erasure code per block, and a codeword symbol is calculated and stored on it (receipted node). Third, for a tenant to migrate his file from one node to another, the nodes in communication authenticates themselves based on pre-measured parameters stored on the log file (TPM). If TPMs accept transfer/communication between communicating nodes, the migration request is allowed without any other hindrance in terms of data security and threats. To archive this, each storage node TPM identifications or provenance information is pre-measured and saved in a log file.

For a node to deliver any storage service as mentioned in categories, it will first measure its authenticity based on pre-stored measurements. And if no matches are found, the service will not be allowed hence system integrity. Lastly, to restore a file from storage nodes, a request is made by a tenant through main control node. In the same sense, when the request is accepted and participating node-to-node authentication process follows, this node continues by decoding codeword symbols from corresponding storage nodes based on encoding settings by erasure code. Therefore, after erasure code decoding process, the main system node merges encoded codeword symbols to derive the original file *f*. In the event that a system node fails, the system generates a new one automatically. After, the new generated storage node requests new codeword symbols from *k* available storage nodes and stores it. Hence, the system node failure is resolved and the original file is computed.

4.1 Security and Confidentiality Model

We look at data confidentiality for both data storage, access and data migration within our cloud model. In this model, when an adversary tries to compromise the system functionality with regard to a specific tenant instance, the adversary tries all storage nodes and tenants. Unfortunately, based on our system architectural TPM-based design, the adversary will try as much iteration as he can to compromise the specific system targets but he will never succeed due to TPM-based security capabilities. However, each storage node will keep all TPM pre-measure log files from neighboring nodes and tenants attached to the same system. From here, a tenant to penetrate into any system attached node, these TPM pre-measured logs must be computed out and compared node-to-node and tenant-to-node. An adversary and tenant have no access to stored logs (log files).

Therefore, the storage cloud design mentioned is trusted if no probabilistic polynomial opportunity for an adversary to detach TPM functions from any of *n* storage nodes on our storage cloud architecture. It implements such that unauthorized node or tenant will never gain access to stored files, all nodes must be considered before by measuring their log parameters and dispersed to each node's TPM through the main control node. Consequently, our model looks at the security of data storage and migration.

5. Secure Storage Cloud Systems Design

Introducing our storage system, we explain the algebraic setting, the robustness supposal, and an erasure code, plus our idea.

Bilinear map: Let G_1 and G_2 be cyclical increasing groupings with a prime order p and $g \in G_1$ be a generator. A map is a bilinear map when it is computationally effective and has the attributes of bilinear and non-degeneracy: as $x, y \in Z_p^*, \hat{e}(g^x, g^y) = \hat{e}(g,g)^{xy}$ and $\hat{e}(g,g)$ is not the identity element in G_2 . Let $Gen(1^\lambda)$ be an algorithm generating $g, \hat{e} G_1, G_2, p$, where λ is the length of p. Let $x \in_R X$ $x =\in_R X$ denote that x is randomly chosen from the set X.

Decisional bilinear Diffie-Hellman supposal: This supposal is that it is computationally impracticable to differentiate the dispersions of $\left(g, g^x, g^y, g^z, \hat{e}(g,g)^{xyz}\right)$ and $\left(g, g^x, g^y, g^z, \hat{e}(g,g)^r\right)$, where $x, y, z, r \in_R Z_p^*$. Therefore, for any probabilistic polynomial time algorithm A, the following is negligible (inλ):

$$\begin{aligned} \Big| \Pr\big[A(g, g^x, g^y, g^z, Q_b) = b : x, y, z, r \in_R Z_p^*, Q_0 \\ = \hat{e}(g,g)^{xyz}; Q_1 \\ = \hat{e}(g,g)^r; b \in_R \{0,1\} \big] - 1/2 \Big] \end{aligned} \tag{1}$$

5.1 Erasure Code Concept

We look at file arena as the increasing cyclical grouping G_2 mentioned earlier. An encoder generates a generator matrix $G = \lfloor g_{i,j} \rfloor$ for $1 <= i <= k, 1 <= j <= n$ as follows: for each row, the encoder randomly selects an entry and randomly sets a value from Z_p^* to the entry. The encoder repeats this step t times with replacement for each row. An input of a row can be selected several times, but it can only be set to one value. The values of the rest inputs are set to zero. Let the data file be $(f_1, f_2, ..., f_k) \in G_2^k$.

The encoding process is to generate $(w_1, w_2, ..., w_n \in G_2^n)$, where $w_1 = f_1^{g,1,j}, f_2^{g,2,j}, ..., f_k^{g,k,j}$ for $1 <= j <= n$. The first step of the decoding process is to

compute the inverse of $k \times k$ sub-matrix K of G. Let K be $\lfloor G_{i,ji} \rfloor$ for $1<=i,\ ji<=k$. And Let $K^{-1} = \lfloor d_{i,j} \rfloor_{1<=i,j<=k}$. The final step of the decoding process is to compute $f_i = w_{j1}^{d1,i}, w_{j2}^{d2,i}, ..., w_{jk}^{dk,i}$ for $1<=i<=k$. Given that a tenant pushes two files f_1 and f_2 into four storage nodes. When the storage nodes N_1 and N_3 are available, and the $k \times k$ sub-matrix K is invertible, a tenant will decode f_1 and f_2 from the codeword symbols w_1, w_2 and the coefficients $(g_{1,1},0), (0, g_{2,3})$ which are stored in the storage nodes N_1 and N_3, a system node failure is resolved.

5.2 Proposed Scheme

We introduce a hardware-based security scheme on storage cloud system nodes. This scheme is trusted if it implements TPM functions to all communicating nodes attached to the storage cloud architecture. Each node on the cloud will host a log file that compose TPM node pre-measured parameters or characteristics as described in **Fig. 3** and **4**. A cloud log file is extended to a node's TPM Platform Configuration Register (PCR).

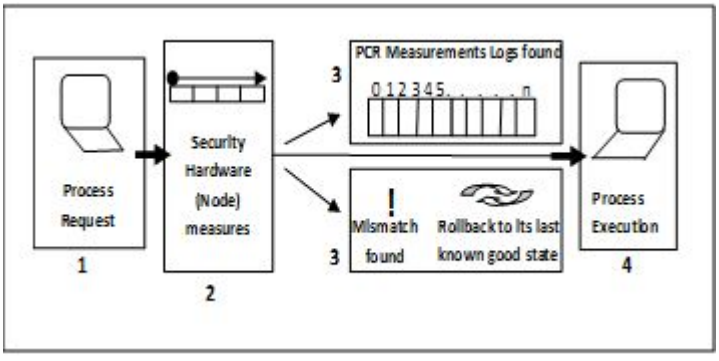

Fig. 3. Log file Storage

According to TPM functionalities underlying on PCRs, an application or a process can use essentially internal memory slots and sealed storage. At TPM boot time, PCR is assigned to its known values, and the only possible means of altering PCR values is by invoking the TPM operation; $\mathrm{PRCExtend(Index, data)}$ where *Index* value is the PCR slot identifier and *Data is* a value carrying TPM's node characteristics (log file). When this operation is invoked, it updates the value of PCR identified by its index with a SH-1 (H) of the previous value of that PCR concatenated with the data provided.

$$PCR_{index} \leftarrow H(PCR_{index}) \| data \tag{2}$$

Data must be 20 bytes and larger DAT A values must be hashed $\mathrm{H(DAT\ A)} \rightarrow \mathrm{data}$ before invoking *PCRExtend*. For a node or any other cloud service to be executed, this log file will be called and compared to both communicating ends.

Given that an adversary tries to enter our system, his source identification will not be recognized as the known log file contents and therefore he will not gain access to our system. For him to gain access, he needs to get into our log file and add or modifies its contents. The access will never succeed due to TPM capabilities.

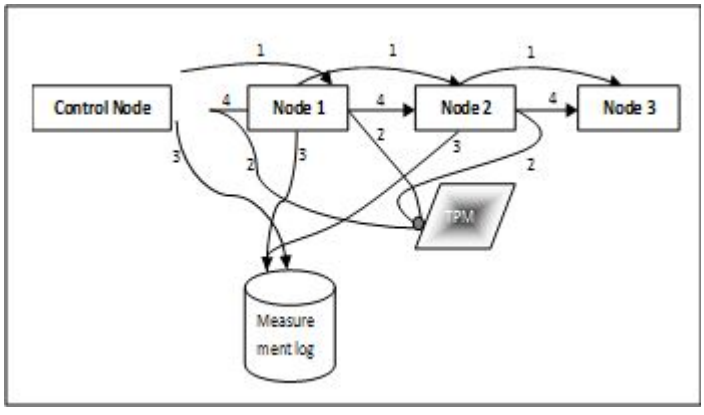

Fig. 4. TPM Node-to-Node Authentication

5.3 A Secure Cloud Data Migration

Referring to the previous section, our storage system is categorized into four categories.

System apparatus: The algorithm $SetUp(1^n)$ derives the system apparatus ∂. A tenant computes his log file content based on his TPM characteristics to derive his credentials that will uniquely allow and identify him from the storage system nodes and services. Running $Gen(1^\lambda)$ will derive $(g, h, \hat{e}, G_1, G_2, p)$, where $\hat{e}: G_1 \times G_1 = G_2$ is a bilinear map, g and h are sources of G_1, and both G_1 and G_2 have the prime order p. Set $\partial = \left(g, h, \hat{e}, G_1, G_2, p, f\right)$, where $f :^*_p X\{0,1\}^* \rightarrow Z^*_p$ and p is a one-way hash function.

Data storage: Given that a tenant A wants to store a file k of blocks $f_1, f_2, \ldots\ldots\ldots, f_k$ with the identifier of the log file *ID*, he distribute both *ID* and k blocks to n storage nodes $N_1, N_2, \ldots\ldots\ldots\ldots, N_n$ of the system. Tenant A pushes each f_i to q randomly selected storage nodes. The storage nodes receive a collection of f with identical source ID from A. However, if any f_i is not receipted to the storage node, the node

enters $f_i = (0,1, ID,1)$ to the group of *k*. Therefore, the particular format of $(0,1, ID,1)$ is a sign for the lost f_i. Also, the storage node performs erasure code function $Encod(\ )$ on the set of *k* and stores the encoded result as codeword symbol. That is to say, $Encod(f_1, f_2, ..., f_k)$. For each f_i, the algorithm randomly chooses a coefficient g_i. Whereas, given that any f_i is lost, the coefficient g_i is initialized to 0. And, let $f_i = (0, \infty_i, \beta, r_i)$. The encoding process is to compute an original codeword symbol f'.

$$\begin{aligned} f' &= \left(0, \prod_{i=1}^{k} \left(\alpha_i^{g}\right)\beta, \prod_{i=1}^{k} \left(y_i^{gl}\right)\right) \\ &= 0, g\sum_{i=1}^{k} g_i y_i, ID, \prod_{i=1}^{k} f_i^{gl}\ \hat{e}(g, ID)\sum_{i=1}^{k} g_i y_i \\ &= 0, g^{r}, ID, w\hat{e}(g, ID)^{r'} \end{aligned} \tag{3}$$

Then, the encoded output is $(f', g_1, g_2, ..., g_k)$.

Data migration: When tenant A requests to migrate his file to another tenant B, he must be known (in the system log file) by the system before. If A is not known by the system log file before, he queries main node for source TPM registration and characteristics measurements to update the system log file. A tenant source behavior must be measured and recorded into the system log file before applying any request to the storage cloud as described in **Fig. 2**. Let the identifier of the log file be ID, and a log file be shared to all storage nodes through the main system node (**Fig. 5**). Therefore, using this file a storage node will allow f_i migration operation with its ID. All storage nodes on the system are interconnected via the main node (in the center) as drawn in the **Fig. 5** below. Each node implements TPM capabilities and erasure code operations. This is the main part of our contribution, and each node performs authentication procedures as illustrated in **Fig. 4** above.

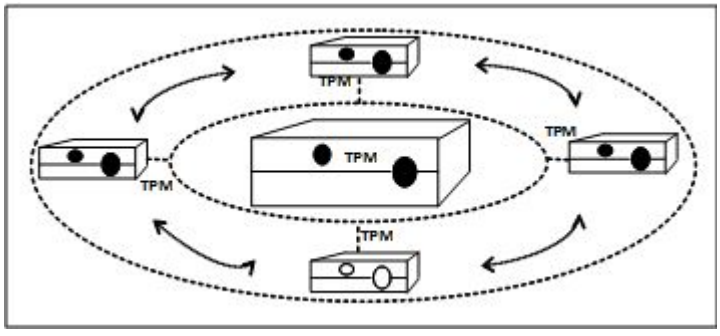

Fig. 5. System Storage Nodes Network

Data restoration: In this category, there are two possible means to restore a file. Firstly, a tenant extracts or restores his own file through an integrity checking process. Given that a request is made to the system attached along with a log file and source ID through the main system node, the main node therefore updates all storage nodes per log file. The system restores codeword symbols from *q* randomly selected storage nodes and then performs decoding process per restored codeword symbol. The decoded restored codeword symbols and log file ID are sent to the requestor. Secondly, the system will accept restoration event on a file that has been migrated from another node. The process undergoes the same as restoring (tenant A) a file as described above.

5.4 The Analysis and Evaluation

Here, we examine storage, computation complications, integrity and security of our storage cloud system. We established the cloud environment using CloudSim simulator on a ThinkPad computer operating on; Ubuntu Linux 14.04, with Inter (R) Core (TM) i5-3230M, 2.60GHz, and 8.00GB RAM. We used TPM API from MIT project to introduce its functionalities on our scheme.

Therefore, the time required for computation process as per the request made from tenant or node depends on communicating TPM and node reliability. Storage capacity and availability are achieved by the main node that locates where and how much storage spaces available for certain storage request initiated by a tenant. We focus on the system security and integrity, the full performance measurements and cloud features remain for further research on the secure multi-cloud scheme.

Let the bit-size of an element in the group G_1 be l_1 and G_2 be l_2. And let coefficients $g_{i,i}$ be randomly chosen from $\{0,1\}g^{l3}$. Then, for storage cost, to store a file of *k* blocks, a storage node N_1 stores a codeword symbol (b, ∞_i, ID, r_i) and the coefficient vector $g_{1,i}, g_{2,i}, \ldots\ldots\ldots\ldots, g_{k,i}$, there are total of $(1+2l_1+l_2+kl_3)$ bits, where $\infty_i, ID \in G_1$ and $r_i \in G_2$.

The average cost for a file bit stored in a storage node is $(1+2l_1+l_2+kl_3)/kl_2$ bits, which is dominated by $l_3 = l_2$ for a sufficiently large *k*.

In practice, small coefficients, i.e., $l_3 << l_2$, reduce the storage cost in each storage node.

6. Conclusion

In this paper, we consider a storage cloud system that consists of storage nodes with TPM and erasure code capabilities. A newly hardware-based security scheme for data integrity and security on the storage systems is presented. It implements data encoding, migration and restoration functions in a dispersed environment. In that, each storage node independently performs data encoding and node-to-node integrity measurements to present

a robust storage cloud system.

Acknowledgement

This work was partly supported by a grant from National Natural Science Foundation Grant NSFC 61272420 and 61472189 through the second author.

References

[1] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, ,C. Wells, B. Zhao and et al, "Oceanstore: An Architecture for Global-Scale Persistent Storage," in *Proc. of Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 190-201, 2000. Article (CrossRef Link)

[2] P. Druschel and A. Rowstron, "PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility," in *Proc. Eighth Workshop Hot Topics in Operating System (HotOS VIII)*, pp. 75-80, 2001. Article (CrossRef Link)

[3] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer and R. Wattenhofer, "Farsite: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," in *Proc. of Fifth Symp. Operating System Design and Implementation (OSDI)*, pp. 1-14, 2002. Article (CrossRef Link)

[4] A. Haeberlen, A. Mislove and P. Druschel, "Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures," in *Proc. Second Symp. Networked Systems Design and Implementation (NSDI)*, pp. 143-158, 2005. Article (CrossRef Link)

[5] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: The Least-Authority Filesystem," in *Proc. of Fourth ACM Int'l Workshop Storage Security and Survivability (StorageSS)*, pp. 21-26, 2008. Article (CrossRef Link)

[6] International Organization for Standardization, "ISO/IEC 11889-1:2009," *ISO.org*, 2013. Article (CrossRef Link)

[7] Trusted Computing Group, "Trusted Platform Module (TPM) Specifications," *Trusted Computing Group*. Article (CrossRef Link)

[8] Trusted Computing Group, "Trusted Platform Module Library," *Trusted Computing Group*. Article (CrossRef Link)

[9] A. Suciu and T. Carean, "Benchmarking the True Random Number Generator of TPM Chips," arXiv:1008.2223, Aug, 2010. Article (CrossRef Link)

[10] Trusted Computing Group, "TPM Main Specification Level 2 (PDF), Part 1 – Design Principles (Version 1.2, Revision 116 ed.)," *Trusted Computing Group*, 2012. Article (CrossRef Link)

[11] Trusted Computing Group, "tspi_data_bind(3) – Encrypts data blob," *Trusted Computing Group*, 2009. Article (CrossRef Link)

[12] Trusted Computing Group, "TPM Main Specification Level 2 (PDF), Part 3 – Commands (Version 1.2, Revision 116 ed.)," *Trusted Computing Group*, 2011. Article (CrossRef Link)

[13] Trusted Computing Group, "TPM – Trusted Platform Module," *IBM*, 2016. Article (CrossRef Link)

[14] US Department of Defense, "Instruction 8500.01 (PDF)," *US Department of Defense*, pp.43, 2014. Article (CrossRef Link)
[15] LUKS, "LUKS Support for storing keys in TPM NVRAM," 2013. Article (CrossRef Link)
[16] RIZZO, L. "Effective erasure codes for reliable computer communication protocols," *ACM SIGCOMM Computer Communication Review*, vol. 27, No. 2, pp. 24–36, 1997. Article (CrossRef Link)
[17] Reed. I. S., and Solomon, G, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, No. 2, pp. 300–304, 1960. Article (CrossRef Link)
[18] H. Abu-Libdeh et al. "Racs", *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, p. 229 – 240, 2010. Article (CrossRef Link).
[19] D.R. Brownbridge, L.F. Marshall and B. Randell, "The Newcastle Connection or Unixes of the World Unite," *Software Practice and Experience*, vol. 12, no. 12, pp. 1147-1162, 1982. Article (CrossRef Link)
[20] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh and B. Lyon, "Design and Implementation of the Sun Network Filesystem," in *Proc. USENIX Assoc. Conf*, 1985. Article (CrossRef Link)
[21] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang and K. Fu, "Plutus: Scalable Secure File Sharing on Untrusted Storage," in *Proc. of Second USENIX Conf. File and Storage Technologies (FAST)*, pp. 29- 42, 2003. Article (CrossRef Link)
[22] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao and J. Kubiatowicz, "Pond: The Oceanstore Prototype," in *Proc. of Second USENIX Conf. File and Storage Technologies (FAST*), pp. 1-14, 2003. Article (CrossRef Link)
[23] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage and G.M. Voelker, "Total Recall: System Support for Automated Availability Management," in *Proc. First Symp. Networked Systems Design and Implementation (NSDI)*, pp. 337-350, 2004. Article (CrossRef Link)
[24] A.G. Dimakis, V. Prabhakaran and K. Ramchandran, "Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes," in *Proc. Fourth Int'l Symp. Information Processing in Sensor Networks (IPSN)*, pp. 111- 117, 2005. Article (CrossRef Link)
[25] A.G. Dimakis, V. Prabhakaran and K. Ramchandran, "Decentralized Erasure Codes for Distributed Networked Storage," *IEEE Trans. Information Theory*, vol. 52, no. 6, pp. 2809-2816, June 2006. Article (CrossRef Link)
[26] H. Y. Lin and W. G. Tzeng, "A Secure Decentralized Erasure Code for Distributed Network Storage," *IEEE Trans. Parallel and Distributed Systems*, vol. 21, no. 11, pp. 1586-1594, Nov. 2010. Article (CrossRef Link)
[27] A. Mehmood, H. Song and J. Lloret, "Multi-Agent based Framework for Secure and Reliable Communication among Open Clouds," *Network Protocols and Algorithms*, Vol. 6, no. 4, pp. 60-76, 2014. Article (CrossRef Link)
[28] E. Baccarelli, N. Cordeschi, A. Mei, M. Panella, M. Shojafar, and J. Stefa, "Energy-Efficient Dynamic Traffic Offloading and Reconfiguration of Networked Data Centers for Big Data Stream Mobile Computing: Review, Challenges, and a Case Study," *IEEE Network*, vol. 30, no. 2, pp. 54-61, 2016. Article (CrossRef Link)
[29] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of Everything: Energy-Efficient Networked Computing Architectures, Research Challenges, and a Case Study," *Access IEE*, Vol.5, pp. 9882-9910, 2017. Article (CrossRef Link)

[30] Y.C. Chen, Y. S. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, ‘‘A measurement-based study of multipath TCP performance over wireless networks,’’ *in Proc. of Conf. Internet Meas. Conf*, pp. 455–468, 2013. Article (CrossRef Link)
[31] F. D. Costa and et al, “Rethinking Internet Things: A Scalable Approach to Connecting Everything,” *New York, NY, USA: Apress*, 2013. Article (CrossRef Link)
[32] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson and D. Song, “Provable Data Possession at Untrusted Stores,” in *Proc. of 14th ACM Conf. Computer and Comm. Security (CCS)*, pp. 598-609, 2007. Article (CrossRef Link)
[33] G. Ateniese, R.D. Pietro, L.V. Mancini and G. Tsudik, “Scalable and Efficient Provable Data Possession,” in *Proc. of Fourth Int’l Conf. Security and Privacy in Comm. Netowrks (SecureComm)*, pp. 1-10, 2008. Article (CrossRef Link)
[34] H. Shacham and B. Waters, “Compact Proofs of Retrievability,” in *Proc. of 14th Int’l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pp. 90-107, 2008. Article (CrossRef Link)
[35] G. Ateniese, S. Kamara and J. Katz, “Proofs of Storage from Homomorphic Identification Protocols,” in *Proc. of 15th Int’l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pp. 319-333, 2009. Article (CrossRef Link)
[36] K.D. Bowers, A. Juels and A. Oprea, “HAIL: A High-Availability and Integrity Layer for Cloud Storage,” in *Proc. of 16th ACM Conf. Computer and Comm. Security (CCS)*, pp. 187-198, 2009. Article (CrossRef Link)
[37] C. Wang, Q. Wang, K. Ren and W. Lou, “Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing,” in *Proc. of IEEE 29th Int’l Conf. Computer Comm. (INFOCOM)*, pp. 525-533, 2010. Article (CrossRef Link)

Emmy Mugisha is a Ph.D (Computer Science and Engineering) student at University of Science and Technology. He attained his MSc (Computer Science and Technology) degree in 2014 from Nanjing University of Information Science and Technology, China; Bachelor degree from University of Rwanda in 2012. He published three papers (1 SCI and 2 EI) in Cloud Computing, Trusted Computing, Information Security, Distributed Computing, Big Data, Provenance Systems and Web Services.

Zhang Gongxuan is a Professor at Nanjing University Science and Technology, School of Computer Science and Engineering. He attained his Ph.D and MSc (Computer Science) degrees in 2005 and 1991 respectively from Nanjing University of Science and Technology, China and Bachelor degree in Computer Science from Tianjin University in 1983. He serves as Deputy Director of ACM Chinese Computer Architecture; IEEE Senior member. He has published more than 60 papers (10 SCI/E and more than 20 EI). His areas of interests are Web Services and Distributed Computing, Trusted Computing and Information Security, Multi-core and High-performance Computing Technology, Networking and Cloud Computing Technology.