International Academic Institute for Science and Technology

International Academic Journal of Science and Engineering
Vol. 2, No. 2, 2015, pp. 211-220.
ISSN 2454-3896

International Academic Journal of Science and Engineering

www.iaiest.com

Enabling Public Auditability and Data Dynamics for checking data integrity in Cloud Computing

Sasan Azhdarpour

Department of Computer, ACECR Institute of higher education kermanshah, Iran

Abstract

The major factor in development limitations of cloud computing is an issue about the data security. The data security protects data versus deviation and unauthorized access. The creation acceptable level of the data security requires an attention of the principles of data security. The most important principle is integrity. integrity ensures authenticity and information accuracy. The data security problems in all types of cloud computing services is an issue, and requires to use of security principles and technical mechanisms safety, in order to solve user's concerns. So far, several methods have been proposed to control the security of cloud computing. But data security in the cloud computing is still a bottleneck. In this study, DICM method is proposed for improving of the data security in cloud computing. The most important features of the proposed method are public auditability, and supporting of dynamic data operations. The proposed method is implemented by using of the simulator CloudSim. Analyzing of the evaluation parameters derived from results of simulation experiments shows that, the process of check integrity has the lowest cost of communication in comparison with other designs, the consumption of network bandwidth is minimizing.

Keywords: Cloud Computing, Data Integrity, Public Auditability, Dynamic Data.

Introduction:

By developing of parallel computing, distributed computing, grid computing, cloud computing model appeared. The most widely accepted current definition for cloud computing is what the National Institute of Standards and Technology (NIST) developed: Cloud Computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. The factors that impel the occurring and development of cloud computing include: the development of grid computing, the appearance of high quality technology in storage and data transportation, and the appearance of Web2.0, especially the development of Virtualization [2]. Improving of productivity, reduction of costs and management facility of servers is one of the most important benefits of this technology [3]. The old dream of computing as a service was realized in the form of cloud computing by using this technology [4], [5]. Today, cloud computing services based on pay peruse model is built and now several large companies including Amazon, Google and Microsoft to provide this service pay.

Cloud computing is offered Internet Cyber space But the Internet is not a controllable entirely place. According to researches which are conducted in [6], in comparison with the efficiency and accessibility, security is always a concern in open system architectures. As a virtual environment cloud computing has its special security threats and these threats are completely different from the threats in physical systems. So the old security policies cannot solve the new cloud computing security issues.

In this study first it reviews existing works in the field of data security, then we will summarize the concepts used in the structure of the proposed method. After that, the proposed method will be described in details. Finally, the performance of the proposed method compared with other existing methods.

Related work

One of the biggest issues with cloud data storage is that of data integrity verification at untrusted servers. For example, the storage service provider, which experiences Byzantine failures occasionally, may decide to hide the data errors from the clients for the benefit of their own. In order to solve the problem of data integrity checking, many schemes are proposed under different systems and security models. Considering the role of the verifier in the models, all the schemes presented before fall into two categories private auditability and public auditability. Although, schemes with private auditability can achieve higher scheme efficiency, public auditability allows anyone, not just the client (data owner), to challenge the cloud server for correctness of data storage while keeping no private information. Then, clients are able to delegate the evaluation of the service performance to an independent third party auditor (TPA), without devotion of their computation resources.

Ateniese et al [7], are the first to consider public auditability in their defined provable data possession model for ensuring possession of files on untrusted storages. In this scheme, they utilize RSA-based homomorphic tags for auditing outsourced data, thus public auditability is achieved. However, Ateniese et al. do not consider the case of dynamic data storage, and the direct extension of their scheme from static data storage to dynamic case may suffer design and security problems. Juels and Kaliski [8] describe a proof of retrievability model, where spot-checking and error-correcting codes are used to ensure both possession and retrievability of data files on archive service systems. Specifically, some special blocks called sentinels are randomly embedded into the data file F for detection purpose, and F is further encrypted to protect the positions of these special blocks. However, like [9] the number of queries a client can perform is also a fixed priori, and the introduction of precomputed sentinels prevents the development of realizing dynamic data updates. In addition, public auditability is not supported in their scheme. Still, the authors only consider static data files. Erway et al. [10], were the first to explore constructions for dynamic provable data possession. They extend the PDP model in [7] to support

provable updates to stored data files using rank-based authenticated skip lists. This scheme is essentially a fully dynamic version of the PDP solution. To support updates, especially for block insertion, they eliminate the index information in the tag computation in Ateniese's PDP model [7] and employ authenticated skip list data structure to authenticate the tag information of challenged or updated blocks first before the verification procedure. However, the efficiency of their scheme remains unclear.

The proposed data integrity checking method

Definitions

Cloud data storage architecture

A representative network architecture for cloud data storage is illustrated in Figure (1). Three different network entities can be identified as follows [7].
Client: an entity, which has large data files to be stored in the cloud and relies on the cloud for data maintenance and computation, can be either individual consumers or organizations.
Cloud Storage Server (CSS): an entity, which is managed by Cloud Service Provider (CSP), has significant storage space and computation resource to maintain the client's data.
Third Party Auditor (TPA): an entity, which has expertise and capabilities that clients do not have, is trusted to assess and expose risk of cloud storage services on behalf of the clients upon request.

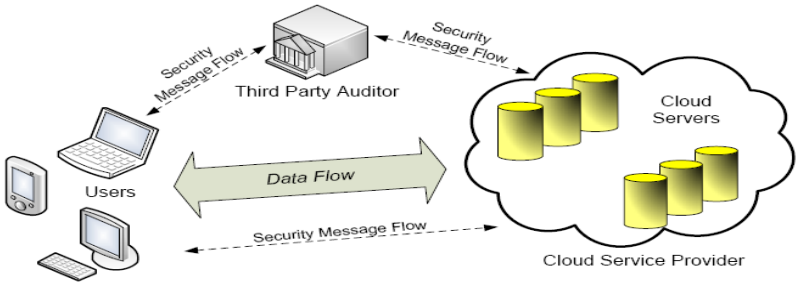


Figure (1) Cloud data storage architecture

Merkle hash tree

A merkle hash tree (MHT) is a well- studied authentication structure which is intended to efficiently and securely prove that a set of elements are undamaged and unaltered [11]. It is constructed as a binary tree where the leaves in the MHT are the hashes of authentic data values. Figure (2) depicts an example of authentication.

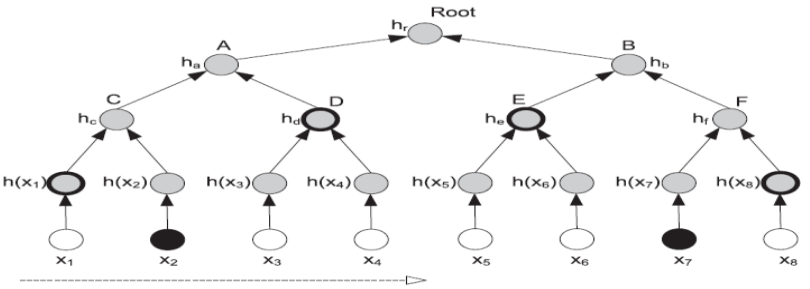


Figure (2) Merkle hash tree authentication of data elements

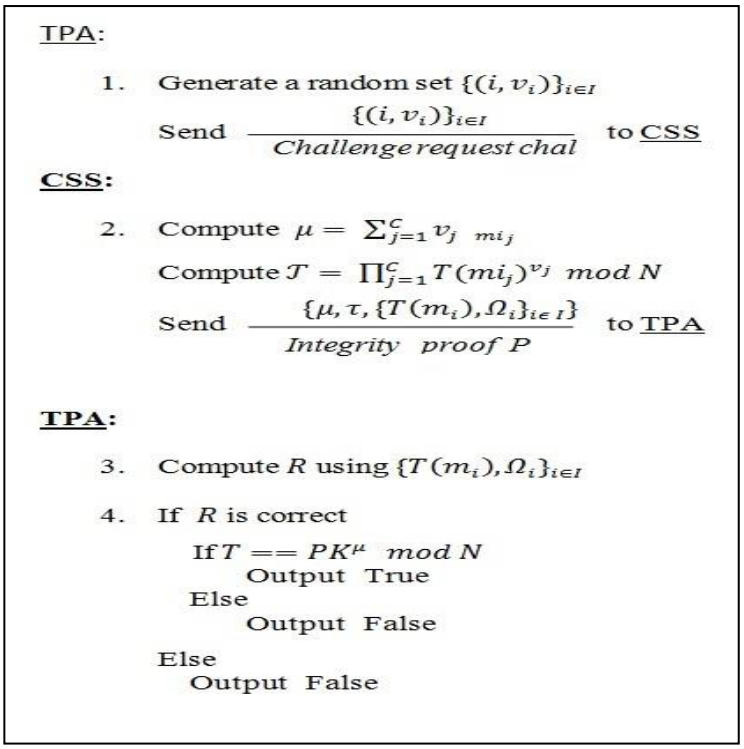
The verifier with the authentic hr requests for $\{x_2 , x_7\}$ and requires the authentication of the received blocks. The prover provides the verifier with the auxiliary authentication information (AAI) $\{\Omega_2 = h(h(x_1), h_d), \Omega_7 = h(h(x_8), h_e)\}$.
The verifiercan then verify x_2 and x_7 by first computing
$h(c) = h(h(x_1) \,||\, h(x_2)), h(f) = h(h(x_7) \,||\, h(x_8)), h(a) = h(h_c || h_d), h(b) = h(h_e || h_f), hr = h(h_a || h_b)$ and then checking if the calculated hr is the same as the authentic one.

Check the integrity of data stored

Client gives public key Pk, $sig_{sk}(T(R))$ and t, to TPA. Now TPA can check integrity file F.Figure (3) shows the process of check data integrity in the proposed method DICM. In step(1), To generate the message *"chal"* the TPA (verifier) chooses a random c-element subset $I = \{s_1, s_2, \dots, s_c\}_{s_1 \le \dots \le s_c}$ of set $[1, n]$ (n=counts of blocks file$F$), For each $i \in I$ the TPA chooses a random element v_i. The message *"chal"* specifies the positions of the blocks to be checked in this challenge phase. The TPA sends the $Chal\ \{(i, v_i)\}_{s_1 \le i \le s_c}$ to the prover (server). In step (2), After receiving the challenge, server runs algorithm $GenProof(.)$. This algorithm receives File F, the set of tags Φ and message $chal$. Then computes two values μ and $\mathcal{T}$.

$$\mu = \sum_{j=1}^{c} v_j mi_j \qquad\qquad \mathcal{T} = \prod_{j=1}^{c} T(mi_j)^{v_j} mod N$$

Here $1 \le i_1, \dots, i_C \le n$ refers to the blocks indexes challenged mi_j , C number of blocks challenged and n the total number of blocks file$F$. $v_j$is also a random value is sent by the client or TPA in message $Chal$. In μ *and* $\mathcal{T}$data blocks challenged and labels which belong to them, have been collected in a single block. The size of the two combined blocks$\mu, \mathcal{T}$ isabout the size of a single block. So there is a very small overhead to send block C. Moreover, in this algorithm a small amount auxiliary authentication information$\{\Omega_i\}_{s_1 \le i \le s_c}$ computed to each block of the set C blocks challenged. Ω_i includes sibling nodes on the path from the leaf $\{h(T(m_i))\}_{s_1 \le i \le s_c}$ to the R rootnode intheMHT. At the end of this step, the server sends proof P to TPA in form of $P = \{\mu, \mathcal{T}, \{T(m_i), \Omega_i\}_{s_1 \le i \le s_c}\}$ .In step (3), TPA after receiving P fromthe server, runs the algorithm $VerifyProof(.)$. In this algorithm, first by using $\{T(m_i), \Omega_i\}_{s_1 \le i \le s_c}$ computes root R in MHT, then is compared with metadata $sig_{sk}(T(R))$ that exists in client's local storage, if it does not equal the output is false, otherwise in step (4) based on the values μ and $\mathcal{T}$, check $\mathcal{T} \stackrel{?}{=} Pk^{\mu} \mod N$ that μ and $\mathcal{T}$ exist in proof P.If so, output is True, otherwise False.



Figure(3) The method of DICM

Dynamic Data Operation with Integrity Assurance

Now we show how our scheme can explicitly and efficiently handle fully dynamic data operations including data modification (M), data insertion (I), and data deletion (D) for cloud data storage. Note that in the following descriptions, we assume that the file F and the hash values Φ have already been generated and properly stored at server. R root created by the client in structure MHT and then signed by the client is stored in local memory .So that the TPA can challenge the correctness of data storage with client's public key and metadata R.

Data Modification: We start from data modification, which is one of the most frequently used operations in cloud data storage. A basic data modification operation refers to the replacement of specified blocks with new ones. Suppose the client wants to modify the ith block m_i tom'_i. This process in DICM is described in Figure (4).

in step (1), based on the new block m'_i, the client generates the corresponding tag $T(m'_i) = Pk^{m'_i}\ mod\ N$, Then, he constructs an update request message $update = \left(M, i, m'_i, T(m'_i)\right)$ and sends to the server, where M denotes the modification operation. In step (2), Upon receiving the request, the server runs algorithm $ExecUpdate(.)$.This algorithm performs the following actions:

1) replaces the block m_i with m'_i and outputs F'
2) replaces the $T(m'_i)$ with $T(m_i)$ and outputs the set of tag Φ'
3) replaces $T(m'_i)$ with $T(m_i)$ in the MHT construction and generates the new root R'

Finally, the server responses the client with a proof for this operation, $P_{update} = (\Omega_i, T(m_i), R')$ where Ω_i is the AAI for authentication of m_i. In step (3), After receiving the proof P_{update} for modification operation from server, the client runs algorithm $Verifyupdate(.)$.In this algorithm, first using $(\Omega_i, T(m_i))$ generates root R in MHT then this value with metadata R (available in local memory client) is compared. If it does not equal the algorithm returns False. Otherwise, in step (4), it is Checked whether the server has done data modification or not? For this purpose, the root R_{new} of MHT base on modified file by using $(\Omega_i, T(m'_i))$ is calculated, then checks $R_{new} \stackrel{?}{=} R'$ if so, the client signed R' in P_{update} and replaces with metadata R (The unmodified version File) that saved in client's local storage. Finally, client runs method data integrity (fig.6), if the output is True then removes m'_i and P_{update} in local storage.

Client:

1. Generate $m'_i, T(m'_i)$
   Send $\dfrac{(M, i, m'_i, T(m'_i))}{Update\ \ request\ \ update}$ to CSS

CSS:

2. Update F, Φ
   Compute R'
   Send $\dfrac{(\Omega_i, T(m_i), R')}{Update\ \ proof\ P_{update}}$ to Client

Client:

3. Compute R using $(\Omega_i, T(m_i))$
   If R is invalid
   Exit
4. Compute R_{new} using $(\Omega_i, T(m'_i))$
   If $R_{new} \stackrel{?}{=} R'$
   Sign R'
   Update R' *s signature*

Figure (4) Dynamic data operation

Data Insertion: Compared to data modification, which does not change the logic structure of client's data file, another general form of data operation, data insertion, refers to inserting new blocks after some specified positions in the data file F. Suppose the client wants to insert block m^* after the ith block m_i . The method procedures are similar to the data modification case (see Figure (4), now m'_i can be seen as

m^*). The difference is that in step (2), The server after receiving a request $update$, runs algorithm $ExecUpdate(.)$.This algorithm performs the following actions:

1) Stores block m^* and adds leaf node $h(T(m^*))$ after that leaf $h(T(m_i))$ in MHT and offers new file F' in the output.
2) Adds $T(m^*)$ to set of tag Φ '
3) Produced new root R' based MHT updated

Continuation of the process data insertion is similar to data modification process.

Data Deletion: data deletion is just the opposite operation of data insertion. For single block deletion, it refers to deleting the specified block and moving all the latter blocks one block forward. Suppose the server receives the update request for deleting block m_i, it will delete m_i from its storage space, delete the leaf node $h(T(m_i))$ in the MHT and generate the new root metadata R'. The details of the method procedures are similar to that of data modification and insertion, which are thus omitted here.

Simulation experiments

The platform of simulation experiment is CloudSim 3.0 which is a simulation platform based on Java. Special users and resources can be generated by rewriting these interfaces. This aligns well with various users and resources of cloud computing. This aligns well with randomness of cloud computing entity action. Therefore, it is feasible to simulate our proposed method by CloudSim. In this method encryption standard algorithms used which consists of Public key RSA, Symmetric encryption AES and hashing algorithm MD5. To implement these algorithms in simulation of proposed method is used OpenSSL0.9.8 library. To precisely simulate the network delay, there is 25~45ms waiting time before a message is sent. The simulation environment is composed of two Entities (data center and client) which Configuration of these entities attributes are asTable1.

Table 1- simulation entities attributes

Entity	Processor Architecture	Operating System	RAM (GB)	Storage(GB)
Data Center	Intel Core 2 3.0 GHz	Linux	16	500
Client	P4 3.0 GHz	Win XP	4	20

Evaluation simulation results of DICM

In Table 3, Data integrity checking schemes are compared with the DICM .

Table 2- Comparisons of different remote data integrity checking schemes

Metric \ Scheme	PDP	POR	DPDP	DICM
Data dynamics	No	No	Yes	Yes
Public auditability	Yes	Yes	No	Yes
Server comp. complexity	O(1)	O(1)	O (log n)	O (log n)
Client comp. complexity	O(1)	O(1)	O (log n)	O (log n)
Comm. Complexity	O(1)	O(1)	O (log n)	O (log n)
Probability of detection	$1 - (\frac{1}{C} - f)$	$1 - (\frac{1}{C} - f)$	$1 - (\frac{1}{C} - f)$	$1 - (\frac{1}{C} - f)$

Public auditability**:** in all data Integrity Checking Schemes except DPDP, Client to challenge the cloud servers for correctness of data storage does not need to explicit retrieval of data blocks. The DICM, can check the integrity of data without retrieving the data blocks. In this way obtained Public auditability.

Data dynamics**:** POR and PDP schemes, Consider only static data. These schemes by using information index i create block tag. So, after inserting or delete the block m_i also other blocks index Changes. Therefore, to insert or delete a block, all other blocks tags will be calculated based on new indices and this is unacceptable over head calculation. So, these schemes do not support dynamic data operation. But, in DPDP dynamic data is considered for the first time. For this purpose, in DICM similar to DPDP information indexes have been removed to create tags and for the block m_i are used $T(m_i)$ as tag. Whereas in PDP and POR, $T(name||i)$ and $T(V||i)$ is used. Accordingly, in DICM and DPDP operations on each data block will not affect to the blocks.

Parameters complexity**:** These parameters are computation server, computation client and communication cost. POR and PDP schemes, Consider only static data. Therefore, constant time achieve for parameters complexity. But, DPDP and DICM are considered to dynamic data. So the time complexity of Parameters is $O(log\ n)$.

Probability of detection**:** this parameter shows the probability of detecting server misbehavior and this parameter value is $1 - (1 - f)^C$ for all schemes **(**where F is the ratio of corrupted blocks and C is the ratio of challenged blocks**).**

In table 2, the results of the simulation tests the data integrity check schemes are shown. Accordingly, the cost of communications PDP is significantly less than other schemes. But it is considerable that the PDP unlike other schemes does not support the dynamics data operation. Increasing the cost of communications in DICM and DPDP is solely for supporting the dynamics data operation. Whereas in both of DPDP and DICM, the client and server computational overhead compared with the PDP has not increased substantially.

Table 3- Comparing the average parameters values of evaluation data integrity checking schemes

Data Integrity Checking Schemes	computation time of client (ms)	computation time of server (ms)	communication cost (KB)
PDP	1251	537	207
DPDP	1825	578	583
DICM	1630	554	516

According to Figure (4) , In all schemes the cost of communication almost grows with increasing the block size linearly that mainly due increase in combined block size of verification $\mu = \sum_{j=1}^{c} v_j mi_j$. As seen in Fig.10, PDP has the lowest cost communication among other schemes. But it is considerable that the PDP does not support the dynamics data operation. On the other hand, DPDP has the most cost communications. But The cost of communication in our DICM scheme, is more than the PDP and it is less than the DPDP.

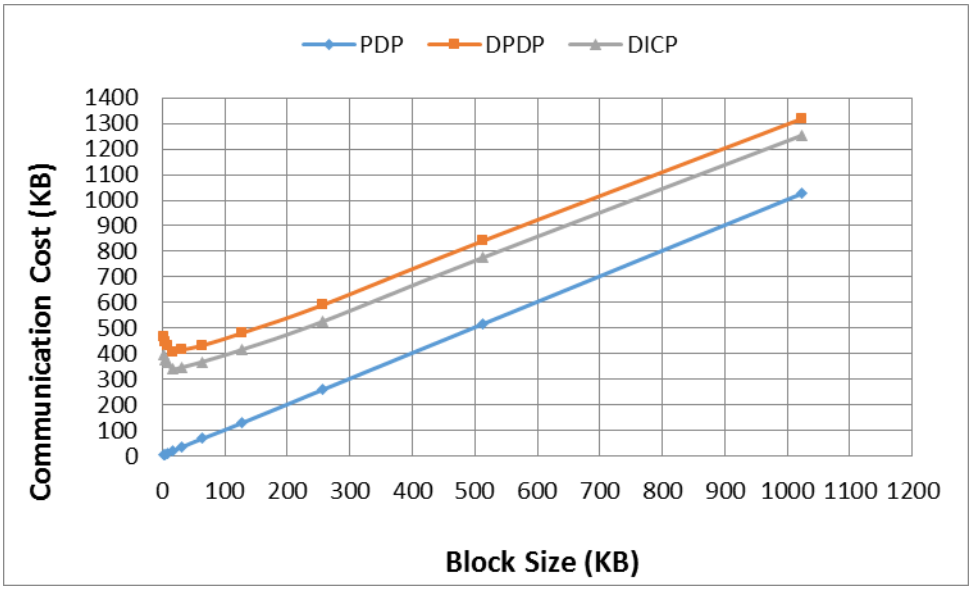


Figure (4) The cost of communications in data integrity check

The results in Fig.10, sh d DPDP schemes can achieve optimal point that it has minimum communication cost. In this case, the lowest communication cost in DICM is 340KB and in DPDP is 405KB.

Conclusion

The proposed method presents two schemes APCC and DICM in order to, protect the confidentiality and integrity, which the most important principles of data security are considered. Performance analysis indicated that DICM is efficient way to check the integrity of outsourced data in data centers in the cloud computing system. On the one hand, with the Public auditability, it is possible that the client is not always concerned about the resources constraints and overhead of stored data integrity checking and the efficiency of storage service assess to at third party in dependent auditor. On the other hand, in DICM the public forms data dynamic operations (modify, delete and insert) is considered. This means that, the index information has been removed to create tags. Accordingly, in DICM operations on each data block will not affect other block. Therefore, does not increase the computation cost. Moreover, DICM has lowest cost communications in comparison with other schemes. Reduce the cost of communications represents less network bandwidth consumption and increase the speed of transfer information in outsourced data integrity checking. DICM scheme reduce the computational overhead on the client side. This aligns well with the idea of cloud computing which allows the user with a platform of limited performance to outsource its computational tasks to some more powerful server.

References:

Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., & Song, D. (2007). *"Provable data possession at untrusted stores"*, In Proceedings of the 14th ACM conference on Computer and communications security, , Alexandria, Virginia, USA, pp. 598-609.

Ateniese, G., Di Pietro, R., Mancini, L. V., &Tsudik, G. (2009). *"Scalable and efficient provable data possession"*, In Proceedings of the 4th international conference on Security and privacy in communication networks, Istanbul, Turkey, pp. 118-129.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., &Brandic, I. (2009). *"Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility"*, Future Generation computer systems, Vol. 25, No. 6, pp. 599-616.

Erway, C., Küpçü, A., Papamanthou, C., &Tamassia, R. (2009). *"Dynamic provable data possession"*, In Proceedings of the 16th ACM conference on Computer and communications security, Chicago, Illinois, USA, pp. 213-222 .

Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., ...&Stoica, I. (2009). *"Above the clouds: A Berkeley view of cloud computing"*, Department Electrical Engineer and Computer Sciences, University of California, Berkeley, Vol. 28.

Juels, A., &Kaliski Jr, B. S. (2007). *" PORs: Proofs of retrievability for large files"*, In Proceedings of the 14th ACM conference on Computer and communications security, , Alexandria, Virginia, USA, pp. 584-597.

Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., & Pu, C. (2007). "*An analysis of performance interference effects in virtual environments"*, In the IEEE International Symposium on Performance Analysis of Systems & Software, San Jose, CA, USA, pp. 200-209.

Kwon, E. K., Cho, Y. G., &Chae, K. J. (2001). "Integrated transport layer security: end-to-end security model between WTLS and TLS". In Proceedings of 15th International Conference on Information Networking, Beppu City, Oita, pp. 65-71.

M Mohamed, E., S Abdelkader, H., & El-Etriby, S. (2013). "Data Security Model for Cloud Computing". In proceedings of the 12th International Conference on Networks, Seville, Spain, pp. 66-74.

Mell, P., Grance, T. (2009). *"The NIST definition of cloud computing, National Institute of Standards and Technology"*, Information Technology Laboratory, Vol. 15, No. 10, pp. 107-347.

Zhang, S., Zhang, S., Chen, X., &Huo, X. (2010)., In *" Cloud computing research and development trend"* proceedings of the 2nd international conference on future networks, Sanya, China, , pp. 93-97