ISSN:2231-0711
www.ijcset.net

Ensuring Privacy and Preventing Silent Data Corruptions in Cloud

R.Sivakami[1], Dr.R.Saravanan[2], N.Pandeeswari[3]
[1,3]*Dept of IT, PSNACET, Dindigul, Tamilnadu, India.*
[2]*Dept of CSE, PSNACET, Dindigul, Tamilnadu, India.*

[1]sivakami81@yahoo.co.in,[2]hodcse@psnacet.edu.in,[3]ponds_it@yahoo.co.in

***Abstract*— Cloud storage systems provide storage service on the internet. Since cloud is accessed via internet, data stored in clouds should remain private. Towards privacy and correctness of the data in cloud, several intrusion detection techniques, authentication methods and access control policies are being used. But they suffer from storage overhead. To minimize this overhead, several designs are currently using erasure codes. Now, the current research on cloud storage systems focuses on efficiency and robustness. But here, the major challenge is how to prevent silent data corruptions during reconstruction of data. This paper uses secure decentralized erasure-coded storage systems to guarantee a better privacy, efficiency and robustness and proposes a method that can reconstruct data in clouds by preventing silent data corruptions.**

***Keywords*-cloud, privacy, silent data corruptions**

I. INTRODUCTION

Internet is a public environment that anyone can freely access. Cloud is a distributed network of storage systems storing data reliably over a very long period of time. Security is essential in cloud because cloud is a multi-tenant environment with more than one company sharing the same cloud service provider. If we use security principles of cloud service provider, our security in the cloud will be as good as, or better, than our current security in most cases. Typically, the level of security we get will be designed to meet the needs of the most risky client in the cloud. But when others have an interest in the contents of our data, privacy is compromised. So it is important to consider the privacy issue of the stored information of the users. Silent Data Corruption occurs when incorrect data is delivered by a computing system to the user without any error being logged. Users need to get correct answers and the resulting data/output needs to have its integrity maintained. So in addition to ensuring privacy in clouds, it is equally important to deal with silent data corruptions in clouds.

The remainder of the paper is organized as follows: In Section 2, we give the existing models to achieve security and privacy in clouds. In Section 3, privacy-preserving public auditing scheme is explained and in section 4, we propose a secure decentralized erasure codes for ensuring privacy in clouds. In Section 5, silent data corruption and its counter-measure are addressed with an efficient adaptive data reconstruction method. Finally, Section 6 concludes our work.

II. EXISTING MODELS

This section explains various methods currently used to achieve security and privacy where data are available on cloud systems. The usual approach to retaining control of data requires the encryption of all cloud data.

A. Identity-Based Authentication for Cloud Computing

For providing authentication of both users and services, SSL Authentication Protocol (SAP), was early used in cloud computing. But complications arise in both computation and communication. So based on the identity-based hierarchical model for cloud computing (IBHMCC), a new method is proposed to achieve security in cloud computing using encryption and signature schemes [2].

Steps:

1. C→ S: ClientHello(n_C, ID, specification$_C$)
ClientHelloDone
2. S→C: ServerHello(n_S, ID, specifications$_S$)
ServerKeyExchange ($E_{P_C}[F_{CS}]$)
IdentityVerify ($Sig_{S_S}[M]$)
ServerHelloDone
3. C→S : ClientFinished

In step (1), the client C sends a ClientHello message to server S. The message contains a random number n_C, the session identifier ID and specification$_C$.

In step (2), the server S responds to client C with a ServerHello message. The message contains a new random number n_S, the session identifier ID and specification$_S$. Then C chooses a pre-master secret F_{CS} and encrypts it with the public key P_C. The ciphertext is transmitted to C as ServerKeyExchange message. Then S generates a signature $Sig_{S_S}[M]$ as the IdentityVerify message to forward to C.

In step (3), C verifies the signature $Sig_{S_S}[M]$ with the help of ID_S. Pass of verification means S is the valid owner of ID_S. This completes authentication from S to C. Then C decrypts the $E_{P_C}[F_{CS}]$ with its private key S_C. The correct decryption indicates C is the valid owner of ID_C.

B. Secure Role based Data Access Control in Cloud Computing

This method [5] associates a set of attributes with each data file and assigns an expressive access structure to each user. This access structure is defined as a unique logical expression of attributes to reflect the scope of data files that the user is allowed to access. As the logical expression can represent any desired data file set, finegrainedness of data access control is achieved. To enforce these access structures, a public key component for each attribute is defined. Data files are encrypted using public key components corresponding to their attributes. User secret keys are defined to reflect their access structures so that a user is able to decrypt a ciphertext if and only if the data file attributes satisfy his access structure. But the challenge with this approach is finegrainedness, data confidentiality, and scalability may not be achieved simultaneously.

C. Virtualization Intrusion Tolerance System

To address the problem of intrusion tolerance of cloud computing platform and sensitive data, a virtualization intrusion tolerance system is constructed based on cloud computing by researching on the existing virtualization technology, and a method of intrusion tolerance is proposed to protect sensitive data in cloud data center [4]. This method allows the system to tolerate F faulty replicas in N=2F+1 replicas and ensure that only F+1 active replicas to execute during the intrusion-free stage. The remaining replicas are all put into passive mode, which significantly reduces the resource consuming in cloud platform.

D. Erasure coded Storage Systems

An erasure code provides redundancy without the overhead of strict replication. Erasure codes divide an object into m fragments and recode them into n fragments, where n>m. We call r=m/n <1 the rate of encoding. The key property of erasure codes is that the original object can be reconstructed from any m fragments. Erasure coding in a malicious environment requires the precise identification of failed or corrupted fragments. Without the ability to identify corrupted fragments, there is potentially a factorial combination of fragments to try to reconstruct the block. As a result, the system needs to detect when a fragment has been corrupted and discard it. A secure verification hashing scheme can serve the dual purpose of identifying and verifying each fragment. It is necessarily the case that any correctly verified fragments can be used to reconstruct the block.

E. Decentralized Erasure Coded storage systems

Decentralized erasure codes are used in distributed networked storage [7]. They are random linear codes over a finite field F_q with a specific randomized structure on their generator matrix. Each data packet D_i is seen as a vector of elements of a finite field f_i. This system of coding has a set of data nodes V_1 with $|V_1| = k$ and storage nodes V_2 with $|V_2| = n$. Towards the construction of a bipartite graph that corresponds to the creation of a decentralized erasure code, every data node $i \in V_1$ is assigned a random set of storage nodes N(i). This set is created as follows: a storage node is selected uniformly and independently from V_2 and added in N(i) and this procedure is repeated d(k) times. Therefore N(i) will be smaller than d(k) if the same storage node is selected twice. $N(j) = \{i \in V_1 : j \in N(i)\}$ denotes the set of data nodes that connect to a storage node. Each storage node will create a random linear combination of the data nodes it is connected with: $S_j = \sum_{\forall i \in N(j)} f_{ij} D_i$ where the coefficients f_{ij} are selected uniformly and independently from a finite field F_q. Each storage node also stores the f_{ij} coefficients, which requires an overhead storage of $N(j)(\log_2(q) + \log_2(k))$ bits.

III. PRIVACY-PRESERVING PUBLIC AUDITING SCHEME

To ensure the correctness and privacy of users' data in the cloud, an effective and flexible distributed scheme is proposed based on the homomorphic token with distributed verification of erasure-coded data[1].

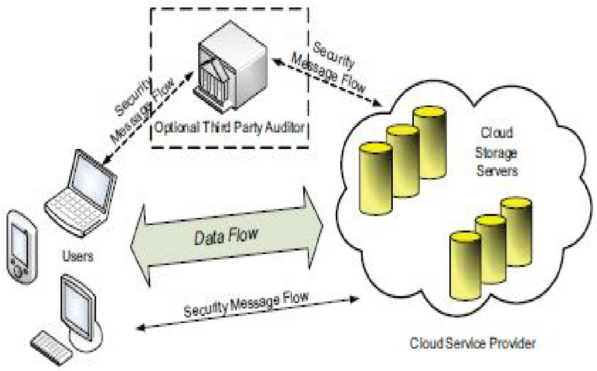

Fig 1: The architecture of cloud data storage service

In Fig 1, Third Party Auditor (TPA) is an entity, which has expertise and capabilities that clients do not have and is trusted to assess and expose risk of cloud storage services on behalf of the clients upon request. Clients may interact with the cloud servers via CSP to access or retrieve their pre-stored data. To ensure privacy TPA should not be allowed to derive users' data content from the information collected during the auditing process.

Steps:

1) TPA retrieves file tag t, verifies its signature.
   If succeeds, it goes to step (2).
   Else, quits.

2) TPA generates a random challenge $chal = \{(i, v_i)\}_{i \in I}$ and sends it to cloud server.

3) Cloud server computes $\mu' = \sum_{i \in I} v_v m_i$ and $\sigma = \prod_{i \in I} \sigma_i^{v_i}$

4) Cloud server randomly selects $r \leftarrow Z_p$ and calculates $\gamma = h(R)$, where $R = e(u,v)^r$. Then it computes $\mu = r + \gamma\mu' \bmod p$.

The storage correctness proof $\{\mu, \sigma, R\}$ is sent to TPA.

TPA computes $\gamma = h(R)$ and then verifies $\{\mu,\sigma,R\}$ *by*

$$R e(\sigma^{\gamma}, g) = e((\prod_{i=s_1}^{s_c} H(W_i)^{v_i})^{\gamma} . u^{\mu}, v)$$

IV. SECURE DECENTRALIZED ERASURE CODED STORAGE SYSTEMS

The methods we have described in section II suffer from storage, computation and communication overhead. So we propose a secure decentralized erasure coded storage systems [3] in this section. It is a combination of data encryption and decentralized erasure codes. Here, data are stored after encryption. Even though the attacker compromises all storage servers, information about the content of data can not be learnt. To achieve this, the decryption key is shared to a set of key servers so that the risk of key leakage is reduced.

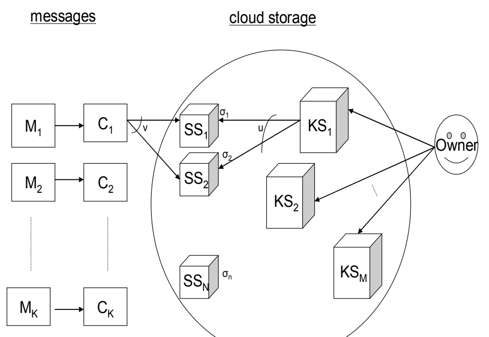

Fig 2: Model of Cloud Storage System

Fig 2 provides an overview of our system. There are k messages M_i, $1 \le i \le k$, to be stored into n storage servers SS_i, $1 \le i \le n$. Let these messages are the segments of a file. For those k messages, we assign a message identifier. Each message M_i is encrypted under the owner's public key pk as $C_i = E(pk, M_i)$ and we consider $pk=g^x$ .Then, each ciphertext is sent to v storage servers, where the storage servers are randomly chosen. Each storage server SS_i combines the received ciphertexts by using the decentralized erasure code to form the stored data σ_i. The owner's secret key sk is shared among m key servers KS_i, $1 \le i \le m$, by a threshold secret sharing scheme so that the key server KS_i holds a secret key share ski. The threshold we assume is t, where $m \ge t \ge k$. To retrieve the k messages, the owner instructs the m key servers such that each key server retrieves stored data from u storage servers and does partial decryption for the retrieved data. Then, the owner collects the partial decryption results, called decryption shares, from the key servers and combines them to recover the k messages.

Steps:

A.. Storing Data:

1) Message Encryption

The owner encrypts k data $M = (M_1 \parallel M_2 \parallel \ldots \parallel M_k)$ via the threshold public key encryption with the same h_{ID}, where $h_{ID} = H(M_1 \parallel M_2 \parallel \ldots \parallel M_k)$ is the identifier. Let the cipher text of be M_i be C_i, where

$$C_i = (\alpha_i, \beta, \gamma_i) = (g^{r_i}, h_{ID}, M_i \tilde{e}(g^x, h_{ID}^{r_i})),$$

$r_i \in_R Z_p, 1 \le i \le k$.

Ciphertext Distribution

For each C_i, the owner randomly chooses v storage servers and a copy of C_i is sent to each of them.

2) Decentralized Encoding

All ciphertexts with the same h_{ID} are grouped into N_j by the storage server SS_j. For each $C_i \in N_j$, SS_j then selects a random coefficient $g_{i,j}$ from Z_p and for each $C_i \notin N_j$, $g_{i,j}$ is set to zero. This task results in a generator matrix $G= [g_{i,j}]$, $1 \le i \le k$, $1 \le j \le n$.

Each SS_j computes

$$A_j = \prod_{C_i \in N_j} \alpha_i^{g_{i,j}} \text{ and } B_j = \prod_{C_i \in N_j} \gamma_i^{g_{i,j}}$$

Then it stores $\sigma_j = (A_j, h_{ID}, B_j, (g_{1,j}, g_{2,j}, \ldots, g_{k,j}))$

B. Retrieving Data:

1) Retrieval Command

The owner sends a command to the m key servers with h_{ID}.

2) Partial Decryption

Each key server KS_i receives at most u stored data σ_j with h_{ID} from the randomly chosen u storage servers. Then each key server KS_i uses its secret key share sk_i to obtain a decryption share of the ciphertext $\zeta_{i,j}$ where $\zeta_{i,j} = (A_j, h_{ID}, h_{ID}^{sk_i}, B_j)$ and sends $\zeta_{i,j}^{\sim}$ to the owner where $\zeta_{i,j}^{\sim} = (A_j, h_{ID}, h_{ID}^{sk_i}, B_j, (g_{1,j}, g_{2,j}, \ldots, g_{k,j}))$.

3) Combining and Decoding

From all received $\zeta_{i,j}^{\sim}$, the owner chooses $\zeta_{i_1,j_1}^{\sim}, \zeta_{i_2,j_2}^{\sim}, \ldots, \zeta_{i_t,j_t}^{\sim}$ and computes $h_{ID}^{sk} = h_{ID}^{f(0)} = h_{ID}^{x}$ by Lagrange interpolation over exponents, where $i_1 \ne i_2 \ne \ldots \ne i_t$ and $S = \{ i_1, i_2, \ldots i_t \}$.

Here $h_{ID}^{x} = \prod_{i \in S} (h_{ID}^{sk_i})^{\prod_{r \in S, r \ne i} \frac{-i}{r-i}}$

After computing h_{ID}^{x}, the owner again selects $\zeta_{i_1,j_1}^{\sim}, \zeta_{i_2,j_2}^{\sim}, \ldots, \zeta_{i_k,j_k}^{\sim}$ with $j_1 \ne j_2 \ne \ldots \ne j_k$ .For all $(i, j) \in \{(i_1, j_1), (i_2, j_2), ..., (i_k, j_k)\}$, the owner decrypts $\zeta_{i,j}$ as w_j, where $w_j = \frac{B_j}{\tilde{e}(A_j, h_{ID}^x)} = \prod_{C_l \in N_j} M_l^{g_{l,j}}$

Then for K = [$g_{i,j}$], 1≤i≤k, $j \in \{j_1, j_2, ..., j_k\}$, the owner computes $K^{-1} = [d_{i,j}], 1 \le i, j \le k$.

If K is invertible, the owner successfully computes M_i with the following computation:

$$w_{j_1}^{d_{1,i}} w_{j_2}^{d_{2,i}} ... w_{j_k}^{d_{k,i}} = M_1^{\sum_{l=1}^{k} g_{1,j_l} d_{l,i}} M_2^{\sum_{l=1}^{k} g_{2,j_l} d_{l,i}} ... M_k^{\sum_{l=1}^{k} g_{k,j_l} d_{l,i}}$$
$$= M_1^{\tau_1} M_2^{\tau_2} ... M_k^{\tau_k}$$
$$= M_i$$

Here, $\tau_r = \sum_{l=1}^{k} g_{r,j_l} d_{l,i} = 1$, if r = i
= 0 , otherwise.

During data retrieval process, the data may be corrupted by detectable and or silent errors. In erasure-coded storage systems, the lost data from detectable errors can be reconstructed by using the redundant data encoded by erasure codes. But problem arises when there exist silent data corruptions.

V. DATA RECONSTRUCTION IN THE PRESENCE OF SILENT DATA CORRUPTIONS

The method explained in section III can be used to reconstruct lost data in the presence of detectable errors. Unfortunately, there exist a large number of silent corruptions where the data is silently corrupted with no indication from the system that an error has occurred. To cope with this challenge, the traditional method is to co-locate metadata with disk data to detect silent corruptions. This metadata can contain extra information, such as checksums or version numbers, which can be used to detect certain types of silent data corruptions. But the metadata method has a remarkable drawback: it requires additional storage overhead to store metadata and thus can involve additional I/O overhead during write operations.

An efficient adaptive data reconstruction method [6] is proposed in this section. It describes how to efficiently reconstruct lost data in the presence of silent data corruptions. We consider a stripe in which f strips are lost or are identified to be corrupted, and there also exist some silently corrupted strips. Suppose there are k^* data strips and m^* parity strips contained in the set of the remaining strips, where $k^* + m^* = k + m - f$. We use D^* and P^* to denote the corresponding data and parity sets, respectively, where $|D^*| = k^*$ and $|P^*| = m^*$.

Steps:

1. Let θ be the number of parity strips in a strip group.
   Set θ = m^*.
2. For each combination P of θ parity strips chosen from P^*, determine whether a consistent strip group containing more than k strips exists in the set of strip groups containing P.
   i) If exists, reconstruct lost strips from this strip group. Detect and correct silently corrupted strips according to this strip group and return true.
   ii) Else, go to step (3)
3. If θ > k – k^* + 1, set θ = θ -1 and go to step (1)
   Else, return false.

The condition to perform this algorithm is f<m. In this algorithm, if true is returned, all the lost strips have been correctly reconstructed, and all the silently corrupted strips have also been detected and corrected; and if false is returned, there exists no consistent strip group containing more than k strips, and the error scenario is then considered to be beyond the ability of the adopted k of (k+m) erasure code to tolerate errors.

The following algorithm determines whether there exists a consistent strip group with a size (k+1) in the set of strip groups containing P.

1. Let $D^* = \{D_0^*, D_1^*, ..., D_{k^*-1}^*\}$, $|P| = \theta$ and $\tau = k - \theta$.
   For a combination D of τ data strips chosen from D^*, we use $i_0, i_1, ..., i_{\tau-2}, i_{\tau-1}$ to denote the subscripts of its τ data strips, respectively.
   Initially set $i_0 = 0,\ i_1 = 1, \ldots, i_{\tau-2} = \tau - 2,$ and $i_{\tau-1} = \tau - 1$.
2. Perform reconstruction operation $D \cup P$.
3. For each data strip with a subscript larger than $i_{\tau-1}$ in D^*, compare its regenerated value with its original value in D^*. If a data strip D_ξ^* exists whose regenerated value is equal to its original value in D^*, return the consistent strip group $D \cup P \cup \{D_\xi^*\}$.
   Else go to next step.
4. If $i_0 = k^* - 1 - \tau$, return NULL.
   Else increase the subscript vector $(i_0, i_1, ..., i_{\tau-1})$ by one in lexicographic order and go back to step (2).

VI. CONCLUSION

In this paper, a secure decentralized erasure code is explained for storing data in a cloud which is a distributed network of storage systems. This method provides privacy, efficiency and robustness in both storage service and key management service, even if all storage services are compromised. And also it is explained how to prevent silent data corruptions from propagating during data reconstruction in the context of erasure coded storage systems.

REFERENCES

[1] Qian Wang, Cong Wang, Kui Ren, Wenjing Louand Jin Li, "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing", IEEE Transactions On Parallel And Distributed Systems, VOL. 22, NO. 5, pp. 847-859, May 2011.

[2] Hongwei Li, Yuanshun Dail, Ling Tian and Haomiao Yang, "Identity –Based Authentication for Cloud Computing", LNCS 5931, pp.157-166, Springer-Verlag Berlin Heidelberg 2009.

[3] Hsiao-Ying Lin and Wen-Guey Tzeng, "A Secure Decentralized Erasure Code for Distributed Networked Storage", IEEE Transactions On Parallel And Distributed Systems, VOL. 21, NO. 11, pp. 1586-1694, November 2010.

[4] Jingyu Wang, xuefeng Zheng and Dengliang Luo, " Sensitive Data Protection Based on Intrusion Tolerance in Cloud Computing", *I.J. Intelligent Systems and Applications*, 2011, 1, 58-66

[5] V.Sathya Preiya, R.Pavithra Dr. Joshi, "Secure Role based Data Access Control in Cloud Computing", International Journal of Computer Trends and Technology- May to June Issue 2011,pp.146-151.

[6] Mingqiang Li and Jiwu Shu, "Preventing Silent Data Corruptions from propagating During Data Reconstruction", IEEE Transactions on Computers, Vol. 59, No.12, pp. 1611-1624, December 2010.

[7] Alexandros G. Dimakis, Vinod Prabhakaran, and Kannan Ramchandran, "Decentralized Erasure Codes for Distributed Networked Storage", IEEE Transactions on Information Theory, Vol. 52, No. 6, pp. 2809-2816 , June 2006.