

# Secure Cloud Storage with Controlled Data Access, Assured Deletion and Data Forwarding

Ch.B.V.Durga<sup>\*</sup>, Mr. Ch. Venkata Narayana<sup>\*\*</sup>, Dr. S. S. S. Reddy<sup>\*\*</sup>

<sup>\*</sup> Research scholar (CSE) Engineering, LBR College of Engineering, Mylavaram, India

<sup>\*\*</sup> Associate Professor (CSE), LBR College of Engineering, Mylavaram, India

<sup>\*\*</sup> HOD (CSE) LBR College of Engineering, Mylavaram, India

**Abstract-** A cloud storage system, consisting of a collection of storage servers, provides long-term storage services over the Internet. Storing data in a third party's cloud system causes serious concern over data confidentiality and to reduce data management costs. However, we must provide security guarantees for the outsourced data. We design and implement a secure cloud storage system that achieves fine-grained, policy-based access control file assured deletion and secure data forwarding to specified users. It associates outsourced files with file access policies, assuredly deletes files to make them unrecoverable to anyone upon revocations of file access policies. To achieve such security goals, a set of cryptographic key operations that are maintained by a separate key server(s) or manager(s). We propose a threshold proxy re-encryption scheme and integrate it with a decentralized erasure code such that a secure distributed storage system is formulated. The distributed storage system not only supports secure and robust data storage and retrieval, but also lets a user forward his data in the storage servers to another user without retrieving the data back.

**Index Terms-** Decentralized erasure code, proxy re-encryption, threshold cryptography, secure storage system, access control, assured deletion, cloud storage

## I. INTRODUCTION

Cloud computing providers offer their services according to several fundamental models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) where IaaS is the most basic and each higher model abstracts from the details of the lower models [10]. Cloud storage is a new business solution for remote backup, as it offers an abstraction of infinite storage space for clients to host data backups in a pay-as-you-go manner. It helps organizations and government agencies significantly reduce their financial overhead of data management, since they can now archive their data backups remotely to third-party cloud storage providers rather than maintain data centers on their own. Many services like email, Net banking, E-commerce etc... are provided on the Internet such that clients can use them from anywhere at any time. Cloud computing is a concept that treats the resources on the Internet as a unified entity, a cloud. Clients just use services without being concerned about how computation is done and storage is managed. In this paper, we focus on designing a cloud storage system for robustness, confidentiality, file access controls and functionality. A cloud storage system is considered as a large

scale distributed storage system that consists of many independent storage servers.

The major requirement for storage systems is data robustness. There have been many proposals of storing data over storage servers. One way to provide data robustness is to encode a message of  $k$  symbols into a codeword of  $n$  symbols by erasure coding [3]. To store a message, each of its codeword symbols is stored in a different storage server. A storage server failure corresponds to an erasure error of the codeword symbol. As long as the number of failure servers is under the tolerance threshold of the erasure code, the message can be recovered from the codeword symbols stored in the available storage servers by the decoding process. This provides a tradeoff between the storage size and the tolerance threshold of failure servers. After the message symbols are sent to storage servers, each storage server independently computes a code-word symbol for the received message symbols and stores it. This finishes the encoding and storing process. The recovery process is the same.

To provide strong confidentiality for messages in storage servers, a client can encrypt messages by a cryptographic method before applying an erasure code method to encode and store messages. There are three problems in the above straightforward integration of encryption and encoding. First, the user has to do most computation and the communication traffic between the user and storage servers is high. Second, the user has to manage his cryptographic keys. If the user's device of storing the keys is lost or compromised, the security is broken. In this paper, we address the problem of forwarding data to another user by storage servers directly under the command of the data owner. The tight integration of encoding, encryption, and forwarding makes the storage system efficiently meet the requirements of data robustness, data confidentiality, and data forwarding. Accomplishing the integration with consideration of a distributed structure is challenging. We consider the system in a more general setting than previous works. This setting allows more flexible adjustment between the number of storage servers and robustness. Assume that there are  $n$  distributed storage servers and  $m$  key servers in the cloud storage system. A message is divided into  $k$  blocks and represented as a vector of  $k$  symbols [3].

One of the other major problems is files accessing. File access policies are used to control unauthorized access. (e.g., time expiration, read/write permissions of authorized users), such that data files are accessible only to clients who satisfy the file access policies. In addition in this paper, generalizes time-based file assured deletion

Our contributions are as follows:

- **Policy based access control and file assured deletion**
- **Secure data forwarding to specified users**

## II. RELATED WORK

### *Policy-Based Deletion*

We now generalize time-based deletion [1][2] to policy-based deletion as follows: we associate each file with a single atomic file access policy (or policy for short), or more generally, a Boolean combination of atomic policies. Each (atomic) policy is associated with a control key, and all the control keys are maintained by the key manager. Suppose now that a file is associated with a single policy. Then similar to time-based deletion, the file content is encrypted with a data key, and the data key is further encrypted with the control key corresponding to the policy. When the policy is revoked, the corresponding control key will be removed from the key manager. Thus, when the policy associated with a file is revoked and no longer holds the data key and hence the encrypted content of the file cannot be recovered with the control key of the policy. In this case, we say the file is assuredly deleted. The main idea of policy-based deletion is to delete files that are associated with revoked policies.

The definition of a policy varies across applications. In fact, time-based deletion is a special case under our frame-work. In general, policies with other access rights can be defined. To motivate the use of policy-based deletion, let us consider a scenario where a company outsources its data to the cloud. We consider four practical cases where policy-based deletion will be useful [4].

- Storing files for tenured employees.
- Storing files for contract-based employees.
- Storing files for a team of employees.
- Switching a cloud provider.

### *Distributed Storage Systems*

At the early years, the Network-Attached Storage (NAS) and the Network File System (NFS) provide extra storage devices over the network such that a user can access the storage devices via network connection. Afterward, many improvements on scalability, robustness, efficiency, and security were proposed.

A decentralized architecture for storage systems offers good scalability. One way to reduce the expansion rate is to use erasure codes to encode messages. A message is encoded as a codeword, which is a vector of symbols, and each storage server stores a codeword symbol. A storage server failure is modeled as an erasure error of the stored codeword symbol. Random linear codes support distributed encoding, that is, each codeword symbol is independently computed. To store a message of  $k$  blocks, each storage server linearly combines the blocks with randomly chosen coefficients and stores the codeword symbol and coefficients. To retrieve the message, a user queries  $k$  storage servers for the stored codeword symbols and coefficients and solves the linear system. The case that  $n = ak$  for a fixed constant  $a$ . They showed that distributing each block of a message to  $v$  randomly chosen storage servers is enough to have a probability

$1 - k/p - o(1)$  of a successful data retrieval, where  $v = b \ln k$ ,  $b > 5a$ , and  $p$  is the order of the used group. The sparsity parameter  $v = b \ln k$  is the number of storage servers which a block is sent to. The larger  $v$  is, the communication cost is higher and the successful retrieval probability is higher. The system has light data confidentiality because an attacker can compromise  $k$  storage servers to get the message.

A secure decentralized erasure code for the networked storage system [16] in addition to storage servers, their system consists of key servers, which hold cryptographic key shares and work in a distributed way. In their system, stored messages are encrypted and then encoded. To retrieve a message, key servers query storage servers for the user. As long as the number of available key servers is over a threshold  $t$ , the message can be successfully retrieved with an overwhelming probability. One of their results shows that when there are  $n$  storage servers with  $n = ak \sqrt{k}$ , the parameter  $v$  is  $b \sqrt{k} \ln k$  with  $b > 5a$ , and each key server queries 2 storage servers for each retrieval request, the probability of a successful retrieval is at least  $1 - k/p - o(1)$ .

### *A. Proxy Re-Encryption Schemes*

Proxy re-encryption schemes [3][5],  $PK_A$  to a new one under another public key  $PK_B$  by using the re-encryption key  $RK_{A \rightarrow B}$ . The server does not know the plaintext during transformation. Our work further integrates encryption, re-encryption, and encoding such that storage robustness is strengthened.

Type-based proxy re-encryption schemes [8] provide a better granularity on the granted right of a re-encryption key. A user can decide which type of messages and with whom he wants to share in this kind of proxy re-encryption schemes. In a key-private proxy re-encryption scheme, given a re-encryption key, a proxy server cannot determine the identity of the recipient. This kind of proxy re-encryption schemes provides higher privacy guarantee against proxy servers.

Let  $G_1$  and  $G_2$  be cyclic multiplicative groups with a prime order  $p$  and  $g \in G_1$  be a generator. A map  $\tilde{e} : G_1 \times G_1 \rightarrow G_2$  is a bilinear map if it is efficiently computable and has the properties of bilinearity and no degeneracy: for any  $x, y \in Z_p^*$ ,  $\tilde{e}(g^x, g^y) = \tilde{e}(g, g)^{xy}$  and  $\tilde{e}(g, g)$  is not the identity element in  $G_2$ . Let  $\text{Gen}(1^\lambda)$  be an algorithm generating  $(g, \tilde{e}, G_1, G_2, p)$ , where  $\lambda$  is the length of  $p$ . Let  $x \in R_X$  denote that  $x$  is randomly chosen from the set  $X$ .

Decisional bilinear Diffie-Hellman assumption. This assumption is that it is computationally infeasible to distinguish the distributions  $(g, g^x, g^y, g^z, \tilde{e}(g, g)^{xyz})$  and  $(g, g^x, g^y, g^z, \tilde{e}(g, g)^r)$ , where  $x, y, z, r \in R_{Z_p^*}$ .

Threshold proxy re-encryption scheme with multiplicative homomorphic property. An encryption scheme is multiplicative homomorphic if it supports a group operation  $\Theta$  on encrypted plaintexts without decryption  $D(SK, E(PK, m_1) \Theta E(PK, m_2)) = m_1 \cdot m_2$  where  $E$  is the encryption function,  $D$  is the decryption function, and  $(PK, SK)$  is a pair of public key and secret key. Given two coefficients  $g_1$  and  $g_2$ , two message symbols  $m_1$  and  $m_2$  can be encoded to a codeword symbol  $m_1^{g_1} \cdot m_2^{g_2}$  in the encrypted form  $C = E(PK, m_1)^{g_1} \Theta E(PK, m_2)^{g_2} = E(PK, m_1^{g_1} \cdot m_2^{g_2})$ .

A secret key is shared to key servers with a threshold value  $t$  via the Shamir secret sharing scheme, where  $t \geq k$ .

C. Integrity Checking Functionality

Another important functionality about cloud storage is the function of integrity checking. After a user stores data into the storage system, he no longer possesses the data at hand. The user may want to check whether the data are properly stored in storage servers. The concept of provable data possession and the notion of proof of storage are proposed in previous papers. Later, public audit ability of stored data is addressed in previous. Nevertheless all of them consider the messages in the clear text form.

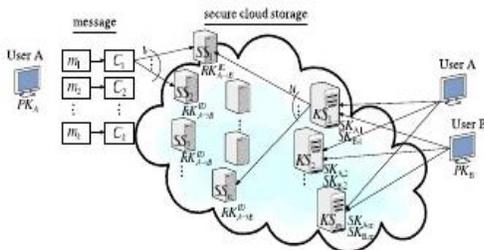
III. SCENARIO

D. System Model

As shown in Fig.1, our system model consists of users, n storage servers  $SS_1, SS_2, \dots, SS_n$ , and m key servers  $KS_1, KS_2, \dots, KS_m$  [3]. Our system consists of **seven** phases: system setup, policy agreement, data storage, data forwarding, data retrieval, policy renewal and assured deletion. These seven phases are described as follows.

In the system **setup phase**, the system manager chooses system parameters and publishes them. Each client A is assigned a public-secret key pair  $(PK_A, SK_A)$ .

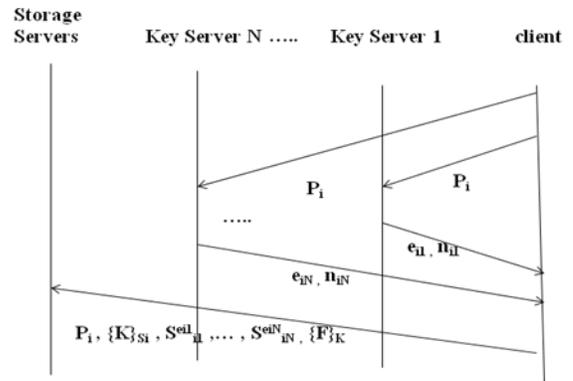
Fig.1. A general system model



User A distributes his secret key  $SK_A$  to key servers such that each key server  $KS_i$  holds a key share  $SK_{A,i}$ ,  $1 \leq i \leq m$ . The key is shared with a threshold  $t$  [3].

In **Policy agreement and upload** phase, mainly focus on two kinds of logical connectives: 1) the conjunction (AND), which means the data is accessible only when every policy is satisfied, and 2) the disjunction (OR), which means if any policy is satisfied, then the data is accessible. Conjunctive policies [4], Suppose that  $F$  is associated with conjunctive policies  $P_1 \wedge P_2 \wedge \dots \wedge P_m$ . To upload  $F$  to the cloud, the user first randomly generates a data key  $K$ , and  $i$  secret keys  $S_1, S_2, \dots, S_i$ . It then sends the following to the cloud:  $\{\{K\}_{S_1}\}_{S_2} \dots \}_{S_i}, S_1^{e_1}, S_2^{e_2}, \dots, S_i^{e_i}$ , and  $\{F\}_K$ . On the other hand, to recover  $F$ , the client generates random number  $R$  and sends  $(S_1 R)^{e_1}, (S_2 R)^{e_2}, \dots, (S_i R)^{e_i}$  to the key server, which then returns  $S_1 R, S_2 R, \dots, S_i R$ . The client can then recover  $S_1, S_2, \dots, S_i$ , and hence  $K$  and  $F$ .

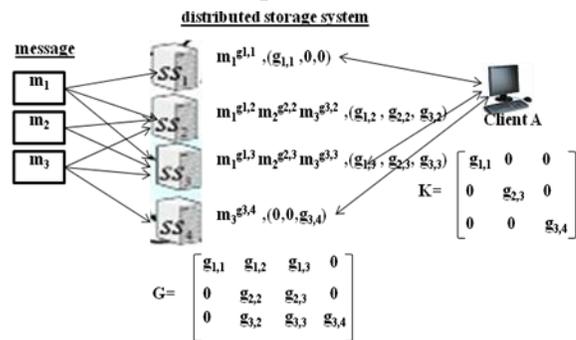
Fig. 2. File upload with multiple key servers



Disjunctive policies, suppose that  $F$  is associated with disjunctive policies  $P_{i1} \vee P_{i2} \vee \dots \vee P_{im}$ . To upload  $F$  to the cloud, the client will send the following:  $\{K\}_{S_1}, \{K\}_{S_2}, \dots, \{K\}_{S_m}, S_1^{e_1}, S_2^{e_2}, \dots, S_m^{e_m}$ , and  $\{F\}_K$ . Therefore, the client needs to compute  $m$  different encrypted copies of  $K$ . On the other hand, to recover  $F$ , we can use any one of the policies to decrypt the file, as in the above operations.

In the **data storage** phase, user A encrypts his message  $M$  and dispatches it to storage servers. A message  $M$  is decomposed into  $k$  blocks  $m_1, m_2, \dots, m_k$  and has an identifier  $ID$ . Client A encrypts each block  $m_i$  into a ciphertext  $C_i$  and sends it to  $v$  randomly chosen storage servers. Upon receiving ciphertexts from a user, each storage server linearly combines them with randomly chosen coefficients into a codeword symbol and stores it. Note that a storage server may receive less than  $k$  message blocks and we assume that all storage servers know the value  $k$  in advance. Erasure coding over exponents [3], consider that the message domain is the cyclic multiplicative group  $G_2$ . An encoder generates a generator matrix  $G = [g_{i,j}]$  for  $1 \leq i \leq k, 1 \leq j \leq n$  as follows: for each row, the encoder randomly selects an entry and randomly sets a value from  $Z_p^*$  to the entry.

Fig. 3. A storage system with random linear coding over exponents

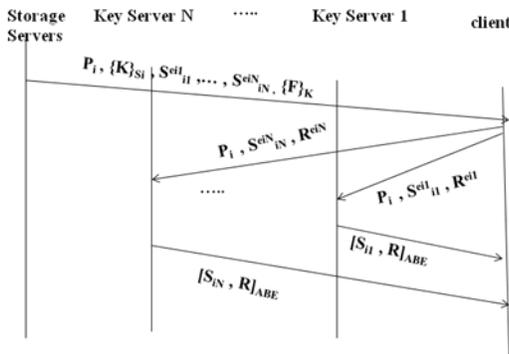


The encoder repeats this step  $v$  times with replacement for each row. The values of the rest entries are set to 0. Let the message be  $(m_1, m_2, \dots, m_k) \in G_2^k$ . The encoding process is to generate  $(w_1, w_2, \dots, w_n) \in G_2^n$ , where  $w_j = m_1^{g_{1,j}} m_2^{g_{2,j}} \dots m_k^{g_{k,j}}$  for  $1 \leq j \leq n$ . The first step of the decoding process is to compute the inverse of a  $k \times k$  sub matrix  $K$  of  $G$ . Let  $K$  be  $[g_{i,j}]$  for  $1 \leq i, j \leq k$ . Let  $K^{-1} = [d_{i,j}]_{1 \leq i, j \leq k}$ . The final step of the decoding process is to compute  $m_i = w_{j_1}^{d_{i,j_1}} w_{j_2}^{d_{i,j_2}} w_{j_k}^{d_{i,j_k}}$  for the 1

$\leq i \leq k$ . An example is shown in Fig.3 Client A stores three messages  $m_1, m_2$  and  $m_3$  into four storage servers. When the storage servers  $SS_1, SS_3$  and  $SS_4$  are available and the  $k \times k$  sub matrix  $K$  is invertible, Client A can decode  $m_1, m_2$  and  $m_3$  from the codeword symbols  $w_1, w_3, w_4$  and the coefficients  $(g_{1,1}, 0, 0), (0, g_{2,3}, 0), (0, 0, g_{3,4})$  which are stored in the storage servers  $SS_1, SS_3$  and  $SS_4$ .

In the **data retrieval** phase, user A requests to retrieve a message from storage servers. The message is either stored by him or forwarded to him. User A sends a retrieval request to key servers. Upon receiving the retrieval request and executing a proper authentication process with user A, Fig. 4 shows the file download operation with multiple key servers. After retrieving the encrypted key shares  $S_{i1}^{ei1}, S_{i2}^{ei2}, \dots, S_{iN}^{eiN}$  from the cloud, the client needs to request each key Server to decrypt a share. For the  $j$ th share  $S_{ij}^{ej}$  ( $j = 1, 2, \dots, N$ ), the client decoys it with a randomly generated number  $R$ , and sends  $S_{ij}^{ej} R^{ej}$  to key Server  $j$ . Then, key Server  $j$  responds the client with  $S_{ij}R$ . It also encrypts the response with ABE [7]. After undecoys, the client knows  $S_{ij}$ . After collecting  $M$  decrypted shares of  $S_{ij}$ , the client decrypts codeword symbols,  $F$  the original message.

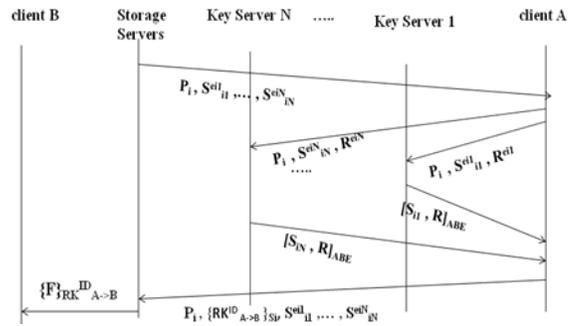
**Fig. 4. Data retrieval with multiple key servers.**



When a storage server fails, a new one is added. The new storage server queries  $k$  available storage servers, linearly combines the received codeword symbols as a new one and stores it. The system is then recovered.

In the **data forwarding** phase, user A forwards his encrypted message with an identifier ID stored in storage servers to user B such that B can decrypt the forwarded message by his secret key. To do so, A uses his secret key  $SK_A$  and B's public key  $PK_B$  to compute a re-encryption key [3]  $RK_{A \rightarrow B}^{ID}$  and then sends  $RK_{A \rightarrow B}^{ID}$  to all storage servers. Each storage server uses the re-encryption key to re-encrypt its codeword symbol for later retrieval requests by B.

**Fig. 5. File forwarding from client A to B**



The re-encrypted codeword symbol is the combination of ciphertexts under B's public key. In order to distinguish re-encrypted codeword symbols from intact ones, we call them original codeword symbols and re-encrypted codeword symbols, respectively.

In **policy renewal** phase, policy renewal means to associate a file with a new policy (or combination of policies). For example, if a user wants to extend the expiration time of a file, then the user can update the old policy that specifies an earlier expiration time to the new policy that specifies a later expiration time.

Policy renewal fully operates on keys, without retrieving the encrypted file from the cloud. The procedures can be summarized as follows [4]:

- Download all encrypted keys from the Storage servers.
- Send them to the key server for decryption.
- Recover the data key.
- Re-encrypt the data key with the control keys of the new Boolean combination of policies.
- Finally, send the newly encrypted keys back to the Storage servers.

**Fig. 6. Policy renewal**

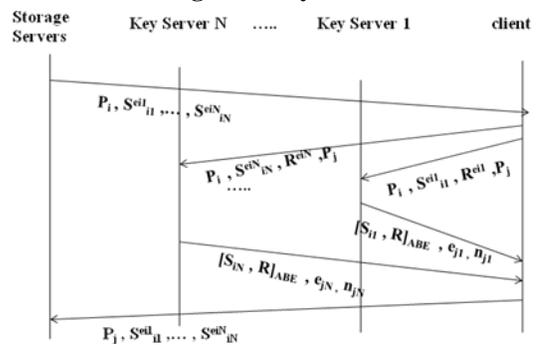


Fig. 6 illustrates this special case of policy renewal. In this case, the client simply sends the decoyed version  $S_i^{ei} R^{ei}$  to the key manager, which then returns  $S_i R$ . The client then recovers  $S_i$ . Now, the client re-encrypts  $S_i$  into  $S_i^{ej} \pmod{n_j}$ , where  $(n_j, e_j)$  is the public control key of policy  $P_j$ , and sends it to the cloud. Note that the encrypted data key  $K$  remains intact.

In **data deletion** phase, to delete a file associated with conjunctive policies, we simply revoke any of the policies (say,  $P_j$ ). Thus, we cannot recover  $S_j$  and hence the data key  $K$  and file  $F$ . On the other hand, to delete a file associated with disjunctive policies, we need to revoke all policies, so that  $S_j^{ej}$  cannot be

recovered for all  $j$ . Note that for any Boolean combination of policies, we can express it in canonical form, e.g., in the disjunction (OR) of conjunctive (AND) policies.

#### *Implementation*

We started our working prototype as web application using Netbeans IDE 6.9.1. Java in Windows OS. Java DB is used as Key storage servers, which is provided by Netbeans IDE. For encryption and decryption processes the inbuilt API's in JDK 6.n was used. In addition, we used LBRCE Storage Servers as our cloud storage. XenServer6.2.0 was used to construct the cloud. JSP was used to create the server pages in our application. The client interacts with the cloud as follows:

#### *Registration:*

The client must register with primary information before going to use the cloud.

#### *Policy Agreement (policy):*

Policy agreement is used to use the cloud storage for the client, which is takes the information about policy and duration of the period.

#### *Upload (policy, file)*

The client encrypts the input file according to the specified policy (or a Boolean combination of policies). Here, the file is encrypted using the 128-bit AES algorithm with the cipher block chaining (CBC) mode. After encryption, the client also appends the encrypted file size (8-bytes long) and the HMAC-SHA1 signature (20-bytes long) to the end of encrypted file for integrity checking in later downloads. It then sends the encrypted file and the metadata onto the cloud.

#### *Forward (Rekey, file, client ID)*

The client A wants to forward a file to client B, client gives the data like recipient id, file name, a special key (rekey)

#### *Download (file)*

The client retrieves the file and policy metadata from the cloud. It then checks the integrity of the encrypted file, and decrypts the file.

#### *Renewal (file, old policy, revised policy)*

The client first fetches the metadata of the given file from the cloud. It updates the metadata with the new policy. Finally, it sends the metadata back to the cloud. Note that the operation does not involve transfer of the input file.

#### *Revocation (policy)*

The client tells the key Servers to permanently revoke the specified policy. All files associated with the policy will be assuredly deleted. If a file is associated with the conjunctive policy combination that contains the revoked policy, then it will be assuredly deleted as well.

#### ACKNOWLEDGMENT

My sincere thanks to our guide Mr.Ch. Venkata Narayana who gave me proper support to do this project. Its special thanks to my Computer Science and Engg. Head Of the Department Dr. S. Sai Satyanarayana Reddy who provide the infrastructure in our college. We are also thankful to Mr. O. Srinivasa Reddy coordinator of our project. We also thank full to Mr. G. Vijay Suresh and staff members for their cooperation.

#### REFERENCES

- [1] R. Perlman, "File System Design with Assured Delete," Proc. Network and Distributed System Security Symp. ISOC (NDSS), 2007.
- [2] R. Geambasu, T. Kohno, A. Levy, and H.M. Levy, "Vanish: Increasing Data Privacy with Self-Destructing Data," Proc. 18th Conf. USENIX Security Symp, Aug. 2009.
- [3] Hsiao-Ying Lin, Member, IEEE, and Wen-Guey Tzeng, Member, IEEE. "A Secure Erasure Code-Based Cloud Storage System with Secure Data Forwarding", JUNE 2012
- [4] Yang Tang, Patrick P.C. Lee, John C.S. Lui, and Radia Perlman "Secure Overlay Cloud Storage with Access Control and Assured Deletion ". NOVEMBER/DECEMBER 2012.
- [5] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage," ACM Trans. Information and System Security, vol. 9, no. 1, pp. 1-30, 2006.
- [6] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," Proc. IEEE Symp. Security and Privacy, May 2006.
- [7] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data," Proc. 13th ACM Conf. Computer and Comm. Security (CCS), 2006.
- [8] Tushar A. Rane, Shrishail T. Patil, Anita H. Khade "Secure Data Transfer using Conditional Re-encryption in Cloud"
- [9] " Survey on Security Challenge for Data forwarding in Cloud" (IJERT) Vol. 2 Issue 1, January- 2013
- [10] Cloudcomputing [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing)
- [11] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A Case for Cloud Storage Diversity," Proc. ACM First ACM Symp. Cloud Computing (SoCC).
- [12] ] Dropbox, <http://www.dropbox.com>.
- [13] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246
- [14] W. Stallings, Cryptography and Network Security. Prentice Hall, 2006.
- [15] "Service Oriented Architecture & Web Services". Dr.Raghu Reddy, Mr.Madan Kumar Srinivasan. Seminar program conducted in LBRCE Aug 2013.
- [16] 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.

#### AUTHORS

**First Author** – Ch.B.V.Durga, Research scholar (CSE) Engineering, LBR College of Engineering, Mylavaram, India  
**Second Author** – Mr. Ch. Venkata Narayana, Associate Professor (CSE), LBR College of Engineering, Mylavaram, India  
**Third Author** – Dr. S. S. S. Reddy, HOD (CSE) LBR College of Engineering, Mylavaram, India