Vol.4 Issue 11, Pg.: 18-24
November 2016

INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

A SECURED FRAMEWORK FOR CLOUD REPOSITORY WITH CLUSTERED SERVERS

Ashok kumar. C[1] and Sathyadevi. S[2]

[1]*Department of Information Technology, PSNA College of Engineering & Technology. – Dindigul. India,* Email ID: ashokinfocom@gmail.com
[2]*Department of Electronics and Communication Engineering, PSNA College of Engineering & Technology – Dindigul India*, Email ID: sathyadevi.s@gmail.com

Abstract: - Cloud Computing is a general term used to describe a new class of network based computing that takes place over the Internet, basically a step on from Service Computing, a collection/group of unified and networked hardware, software and Internet infrastructure. These platforms cover the complexity and details of the underlying framework from users and applications by providing very simple graphical interface or API. It is basically the collection of computers on the internet that companies are using to offer their services. One cloud service that is being offered is a revolutionary repository method for your data. From music files to pictures to conscious documents, the cloud invisibly backs up your files and folders and alleviates the potentially endless and costly search for extra repository space. Our structure is fully meshed with Encryption, keeping and recall process. Introduce a boundary delegate re-encryption method and it meshes with decentralized erasure code such that a protected distributed scheme formulated. Evaluate and offer appropriate limitations for the number of copies of a information transmitted to repository servers and the number of repository servers queried by a key server. These restrictions allow more flexible regulation between the number of repository servers and robustness.

Keywords: Clusters, delegate re-encryption, boundary cryptography, safer repository system.

1. Introduction

Data robustness is a major requirement for repository systems. There have been many proposals of keeping data over repository servers [1], [2], [3], [4], [5]. One way to serve data robustness is to replicate a information such that each repository server stores a copy of the message. It is very robust because the information can be recall as long as one repository server survives. Another way is to encode a information of k pattern into a codeword of n pattern by erasure coding. To store a message, each of its codeword patterns is stored in a different repository server. A repository server failure corresponds to an erasure error of the codeword symbol. The number of failure servers is under the tolerance boundary of the erasure code, the information can be recovered from the codeword pattern stored in the available repository servers by the decoding process. This serves a tradeoff between the repository size and the tolerance boundary of failure servers. A decentralized erasure code is an erasure code that independently computes each codeword symbol for a message. Thus, the encoding process for a information can be split into n parallel tasks of generating codeword pattern. A decentralized erasure code is suitable for use in a distributed repository system. After the information patterns are sent to repository servers, each repository server independently computes a code-word symbol for the received information pattern and stores it. This finishes the encoding and keeping process. The recovery process is the same.

Keeping data in a third party's cloud scheme causes serious concern on data confidentiality. In order to serve strong confidentiality for messages in repository servers, a user can encrypt messages by a cryptographic method before applying an erasure code method to encode and store messages. When it wants to use a message, it needs to retrieve the codeword pattern from repository servers, decode them, and then decrypt them by using cryptographic keys. There are three problems in the above linear integration of encryption and encoding. First, the user has to do most calculation and the communication traffic between the user and repository servers is high. Second, the user has to manage his cryptographic keys. If the user's device of keeping the keys is lost or compromised, the security is broken. Finally, besides data keeping and retrieving, it is hard for repository servers to directly support other functions. repository servers cannot directly forward a user's messages to another one. The owner of messages has to retrieve, decode, decrypt and then forward them to another user.

2. Related work

2.1 Distributed Repository Systems

Network-Attached Storage (NAS) [7] and the Network File Scheme (NFS) [8] serve extra wants to share his messages; it sends a re-encryption key to the repository server. The repository server re-encrypts the encrypted messages for the authorized user. Their scheme has data confidentiality and supports the data forwarding function. Our work further integrates encryption, re-encryption, and encoding such that repository robust-ness is strengthened.

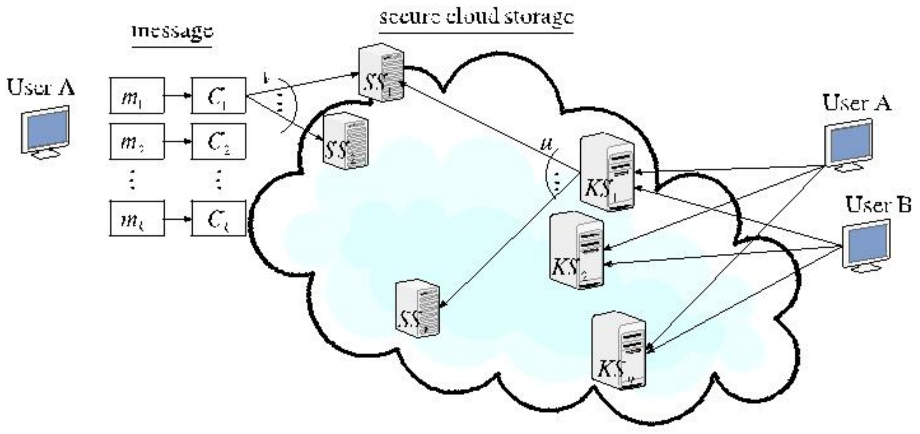


Figure 1.Distributed repository system

2.2 Delegate Re-Encryption Designs

Delegate re-encryption designs are proposed by Mambo and Okamoto [14] and Blaze et al. [15]. In a delegate re-encryption design, a delegate server can transfer a cipher text under a public key A to a new one under another public key B by using the re-encryption key RKA!B. The server does not know the unencrypted text during transformation. Agenise et al. [16] proposed some delegate re-encryption designs and applied them to the sharing function of protected repository systems. In their work, messages are first encrypted by the owner and then stored in a repository server. When a user key server KSI holds a key share SKA. The key is shared with a boundary t. The data forwarding point, user A forwards his encrypted information with an identifier ID stored in repository servers to user B such that B can decrypt the forwarded information by his secret key. , A uses his secret key SKA and B's public key PKB to compute a re-encryption key RKIDA!B and then sends RKIDA!B to all repository servers. Each repository server uses the re-encryption key to re-encrypt its codeword symbol for later retrieval requests by B. The re-encrypted codeword symbol is the combination of cipher texts under B's public key. In order to distinguish re-encrypted codeword pattern from intact ones, we call them original codeword pattern and re-encrypted codeword pattern, respectively.

In the data retrieval point, user A requests to retrieve a information from repository servers. The data 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, each key server KSI requests u randomly chosen repository servers to get codeword pattern and does partial decryption on the received codeword pattern by using the key share SKA;i. Finally, user A combines the partially decrypted codeword pattern to obtain the original information.

2.3 Integrity Checking Functionality

The important functionality about cloud repository is the function of integrity checking. After a user stores data into the repository system, he no longer possesses the data at hand. The user may want to check whether the data are properly stored in repository servers. The concept of provable data possession [20], [21] and the notion of proof of repository [25], [23], [24] are proposed. Public auditability of stored data is addressed in [25]. Nevertheless all of them consider the messages in the clear text form.

3. DESIGNS AND COMPONENTS USED

3.1RSA Algorithm

RSA is an algorithm for a public key encryption system. i.e this is an asymmetric cipher , one person has a private key and gives out a public key, any has the public key can encrypt some information or data then only the person with the private key can read this information, not even the person who encoded the original information with the public key can now decoded this. The idea behind this to prevent" man in the middle attacks".

3.2 Public Key

In cryptography, a public key is a value served by some designated authority as an encryption key that combined with a private key derived from the public key, that can be used to efficient encrypt information and digital signatures. The use of combined public and private key is known as asymmetric cryptography. A scheme for using public keys is called a public key infrastructure (PKI).

3.2 Symmetric Vs Asymmetric Algorithm

When using symmetric algorithms, both parties share the same key for en-and decryption. To serve privacy, this key needs to be stored secret. Once somebody else gets to know the key, it is not safe anymore. Symmetric algorithm has the advantage of not consuming too much computing power. Asymmetric algorithm use pairs of keys. One is used for encryption and the other one for decryption. The decryption key is typically kept secretly, therefore called "Private Key" or "secret key", while the encryption key is spread to all who might need to send encrypted information, therefore called "Public key". Everybody having the public key is able to send encrypted information to the owner of the secret key. The secret key can`t be reconstructed from the public key. Asymmetric algorithm is much slower than symmetric. Therefore, in much application, a merger of both is being used. The asymmetric keys are used for authentication and after this have been successfully done. One or more symmetric keys are generate and exchange using the asymmetric encryption.

3.4 Key Generation

RSA involves a public key and a private key. The public key can be know to everyone and is used for encrypting message. Information encrypted with the public key can only be decrypted using the private key. The keys foe the RSA algorithm are generated the following ways: Generate two large random primes, p and q, of approximately equal size such that their product n=pq is of the required bit length ,e.g. 1024bits.

- Compute n=pq and phi=(p-1)(q-1)
- Choose an integer e,1<e<phi, such that gcd(e,phi)=1
- Compute the secret exponent,1<d<phi, such that ed=1(mod phi)
- The public key is (n,e) and the private key(d,p,q). Keep all the values d,p,q and phi secret, n knows as the modules.
- E is known as the public exponent or encryption exponent or just the exponent.
- D is known as the secret exponent or decryption exponent.

3.5 File Rifting

As shown in the figure 7.2 the input text file is given by the user, according to their wish they splits the file into chunks mentioned by them. For providing effective data exchange and prevention of file from intruders they split the file into number of chunks. The chunks splitting are based on the user given inputs.

3.6 Encryption

Encryption is the conversion of data into a form, called a cipher text that cannot be easily understood by unauthorized people. Splitter files are encrypted using corresponding keys. Here RSA algorithm is used for encryption. The encryption design used for providing high security. This is first step keeping process.

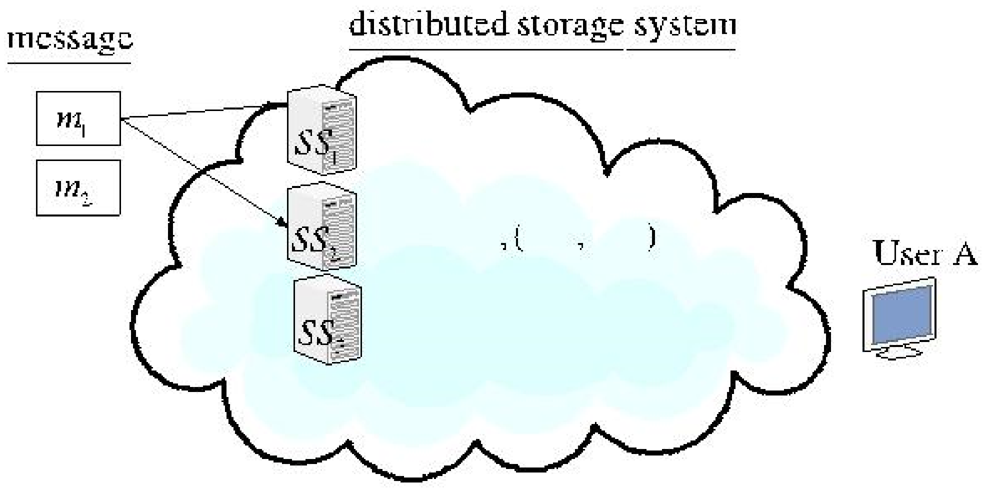


Figure 2 Encryption

3.7 Delegate Re-Encryption Design

Delegate re-encryption designs are cryptosystems which allow third-parties (Proxies) to alter a cipher text which has been encrypted for one party, so that it may be decrypted by another. Crypto scheme consists of three algorithms. That is one for key generation, one for encryption, one for decryption. A delegate server is a server that acts as an intermediate for request from clients seeking resources from other servers. In the proposed scheme encrypted files are re-encrypted using boundary delegate re-encryption design.

3.8 Data Storage on Cloud

As mention earlier the input file which was splitted and encrypted was stored in the cloud server, where the user specified. Cloud was having multiple servers. The re-encrypted chunks are stored in the cloud servers using the ip address of the server name.

3.9 *Data Extraction On Cloud*

Data are downloaded from the cloud, and then the files are re-decrypted by referring the corresponding keys in key server. Re-decrypted files are then decrypted. Decrypted files are joined using joining function. Then we get the original text file.

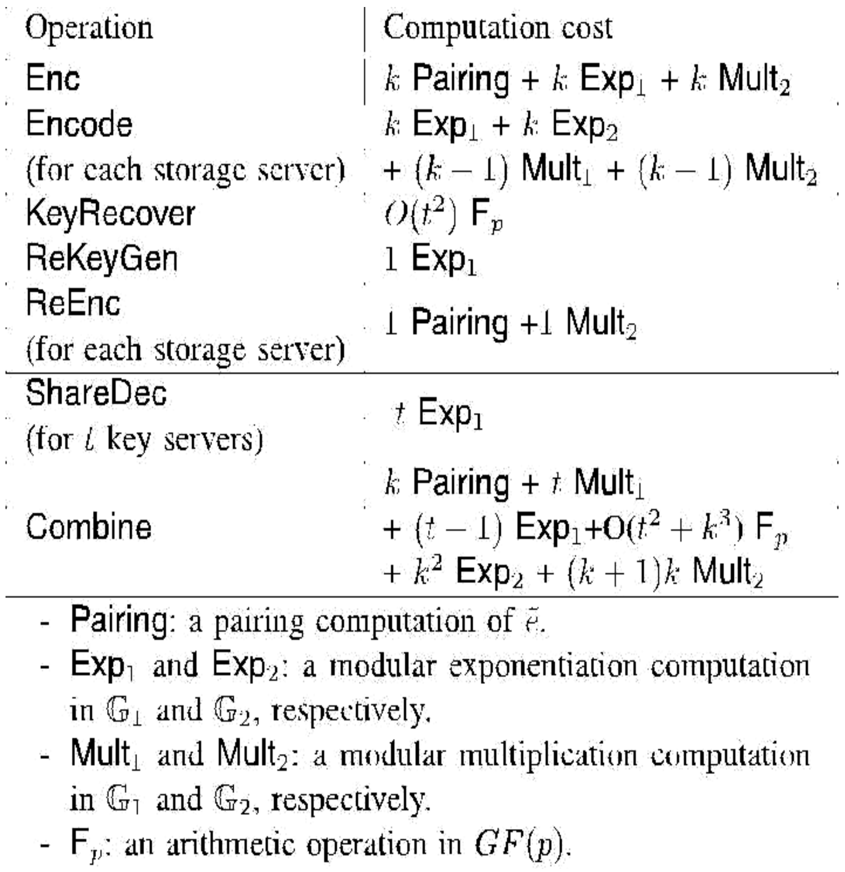
4 RESULTS AND DISCUSSION

4.1 A Protected Cloud Repository Scheme with Protected Forwarding

Data retrieval. There are two cases for the data retrieval point. The first case is that a user A retrieves his own message. When user A wants to retrieve the information with the identifier ID, he informs all key servers with the identity token _. A key server first retrieves original codeword pattern from u randomly chosen repository servers and then performs partial decryption ShareDecð_Þ on every recall original codeword symbol C0. The result of partial decryption is called a partially decrypted codeword symbol. The key server sends the partially decrypted codeword pattern _ and the coefficients to user A. After user A collects replies from at least t key servers and at least k of them are originally from distinct repository servers, he executes Combine on the partially decrypted codeword pattern to recover the blocks m1; m2; . . . ; mk. The second case is that a user B retrieves a information forwarded to him. User B informs all key servers directly. The collection and combining parts are the same as the first case except that key servers retrieve re-encrypted codeword pattern and perform partial decryption Share-Decð_Þ on re-encrypted code-word pattern.

4.2 Analysis

The calculation cost by the number of pairing process, modular exponentiations in G1 and G2, modular multiplications in G1 and G2, and arithmetic process over GF ðpÞ. These processes are denoted as Pairing, Exp1, Exp2, Mult1, Mult2, and Fp, respectively. The cost is summarized in Table 1. Computing an Fp takes much less time than computing a Mult1 or a $\text{Mult}_{2.}$ The methodology of analysis is similar to that in [13] and [6]. However, we consider a different scheme model from the one in [13] and a more flexible parameter setting for n ¼ akc than the settings in [13] and [6]. The difference between our scheme model and the one in [13] is that our scheme model has key servers. In [13], a single user queries k distinct repository servers to retrieve the data. On the other hand, each key server in our scheme independently queries u repository servers. The use of distributed key servers increases the level of key protection but makes the analysis harder.



Operation	Computation cost
Enc	k Pairing + k Exp$_1$ + k Mult$_2$
Encode (for each storage server)	k Exp$_1$ + k Exp$_2$ + $(k-1)$ Mult$_1$ + $(k-1)$ Mult$_2$
KeyRecover	$O(t^2)$ F$_p$
ReKeyGen	1 Exp$_1$
ReEnc (for each storage server)	1 Pairing +1 Mult$_2$
ShareDec (for t key servers)	t Exp$_1$
Combine	k Pairing + t Mult$_1$ + $(t-1)$ Exp$_1$+O(t^2+k^3) F$_p$ + k^2 Exp$_2$ + $(k+1)k$ Mult$_2$

- Pairing: a pairing computation of $\tilde{e}$.
- Exp$_1$ and Exp$_2$: a modular exponentiation computation in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- Mult$_1$ and Mult$_2$: a modular multiplication computation in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- F$_p$: an arithmetic operation in $GF(p)$.

5. CONCLUSIONS

The performance and safety will be high compared to the existing system because of using encryption and re-encryption, for the data kept in the cloud. In olden system the data are stored in single server and no encryption decryption techniques are used. So data loss is high .Cryptographic keys are managed by the user. But in the new system, data are kept in numerous servers; proxy re-encryption scheme is used for providing security and confidentially. Cryptographic keys are maintained by the key server. The user provides the input text file and it was safely stored in the cloud servers by means of doing encryption and proxy re-encryption design.

REFERENCE

[1] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, “Oceanstore: An Architecture for Global-Scale Persistent Storage,” Proc. Ninth Int’l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 190-201, 2000.
[2] A. Haeberlen, A. Mislove, and P. Druschel, “Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures,” Proc. Second Symp. Networked Systems Design and Implementation (NSDI), pp. 143-158, 2005.
[3] Z. Wilcox-O’Hearn and B. Warner, “Tahoe: The Least-Authority File system,” Proc. Fourth ACM Int’l Workshop Storage Security and Survivability (StorageSS), pp. 21-26, 2008.
[4] 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.

[5] 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.

[6] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem," Proc. USENIX Assoc. Conf., 1985.

[7] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable Secure File Sharing on Untrusted Storage," Proc. Second USENIX Conf. File and Storage Technologies (FAST), pp. 29-42, 2003.

[8] S.C. Rhea, P.R. Eaton, D. Geels, H. Weatherspoon, B.Y. Zhao, and J. Kubiatowicz, "Pond: The Ocean store Prototype," Proc. Second USENIX Conf. File and Storage Technologies (FAST), pp. 1-14, 2003.

[9] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G.M. Voelker, "Total Recall: System Support for Automated Availability Management," Proc. First Symp. Networked Systems Design and Implementation (NSDI), pp. 337-350, 2004.

[10] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous Access to Distributed Data in Large-Scale Sensor Net-works through Decentralized Erasure Codes," Proc. Fourth Int'l Symp. Information Processing in Sensor Networks (IPSN), pp. 111-117, 2005.

[11] 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.

[12] M. Mambo and E. Okamoto, "Proxy Cryptosystems: Delegation of the Power to Decrypt Cipher texts," IEICE Trans. Fundamentals of Electronics, Comm. and Computer Sciences, vol. E80-A, no. 1, pp. 54-63, 1997.

[13] M. Blaze, G. Bleumer, and M. Strauss, "Divertible Protocols and Atomic Proxy Cryptography," Proc. Int'l Conf. Theory and Application of Cryptographic Techniques (EUROCRYPT), pp. 127-144, 1998.

[14] 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.

[15] Q. Tang, "Type-Based Proxy Re-Encryption and Its Construction," Proc. Ninth Int'l Conf. Cryptology in India: Progress in Cryptology (INDOCRYPT), pp. 130-144, 2008.

[16] G. Ateniese, K. Benson, and S. Hohenberger, "Key-Private Proxy Re-Encryption," Proc. Topics in Cryptology (CT-RSA), pp. 279-294, 2009.

[17] J. Shao and Z. Cao, "CCA-Secure Proxy Re-Encryption without Pairings," Proc. 12th Int'l Conf. Practice and Theory in Public Key Cryptography (PKC), pp. 357-376, 2009.

[18] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," Proc. 14th ACM Conf. Computer and Comm. Security (CCS), pp. 598-609, 2007.

[19] G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," Proc. Fourth Int'l Conf. Security and Privacy in Comm. Netowrks (Secure Comm.), pp. 1-10, 2008.

[20] H. Shacham and B. Waters, "Compact Proofs of Retrievability," Proc. 14th Int'l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT), pp. 90-107, 2008.

[21] G. Ateniese, S. Kamara, and J. Katz, "Proofs of Storage from Homomorphic Identification Protocols," Proc. 15th Int'l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT), pp. 319-333, 2009.

[22] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, "Hydrastor: A Scalable Secondary Storage," Proc. Seventh Conf. File and Storage Technologies (FAST), pp. 197-210, 2009.

[23] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Calkowski, C. Dubnicki, and A. Bohra, "Hydrafs: A High-Throughput File System for the Hydrastor Content-Addressable Storage System," Proc. Eighth USENIX Conf. File and Storage Technologies (FAST), p. 17, 2010.

[24] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in Scalable Data Routing for Reduplication Clusters,"

[25] A. Shamir, "How to Share a Secret," ACM Comm., vol. 22, pp. 612-613, 1979.