Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Science and Engineering

Elixir Comp. Sci. & Engg. 67 (2014) 21526-21529

ISSN: 2229-712X

A Safe Cloud Storage with Multiple Servers

M. Kannan[1,*], P.K. Kumaresan[2] and S. Palanivel[3]

[1]Department of Information Technology, Mahendra College of Engineering, Mallasamudram, Namakkal Dt, India,
[2]Department of Information Technology, VMKV Engineering College, Vinayaka Missions Univerisity, Periyaseeragapadi, Salem, India.
[3]Department of Computer Science & Engineering, Annamalai University, Chidambaram, Tamilnadu,

ARTICLE INFO

Article history:
Received: 28 December 2013;
Received in revised form:
24 January 2014;
Accepted: 6 February 2014;

Keywords
Decentralized erasure code,
Proxy re-encryption,
Threshold cryptography,
Secure storage system.

ABSTRACT

The security of cloud users, a few proposals have been presented recently. Core objective of using cloud is to provide Security, Scalability, Availability, Performance, and Cost effective. A Safe Cloud Storage to provide confidentiality and fine-grained access control for data stored in the cloud. This system enables the users to enjoy a secure outsourced data services at a minimized security management overhead. Here outsources not only the data but also the security management to the cloud in a trust way. Our system is fully integrates with Encryption , storing and retrival operations. Propose a threshold proxy re-encryption scheme and it integrates with decentralized erasure code such that a secure distributed system is formulated. Analyze and suggest appropriate limitations for the number of copies of a message transmitted to storage servers and the number of storage servers queried by a key server. These restrictions allow more flexible regulation between the number of storage servers and robustness.

© 2014 Elixir All rights reserved

Introduction

Data robustness is a major requirement for storage systems. There have been many proposals of storing data over storage servers [1], [2], [3], [4], [5]. One way to provide data robustness is to replicate a message such that each storage server stores a copy of the message. It is very robust because the message can be retrieved as long as one storage server survives. Another way is to encode a message of k symbols into a codeword of n symbols by erasure coding. 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. 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. A decentralized erasure code is an erasure code that indepen-dently computes each codeword symbol for a message. Thus, the encoding process for a message can be split into n parallel tasks of generating codeword symbols. A decentralized erasure code is suitable for use in a distributed storage system. 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.

Storing data in a third party's cloud system causes serious concern on data confidentiality. In order to provide strong confidentiality for messages in storage servers, a user can encrypt messages by a cryptographic method before apply-ing an erasure code method to encode and store messages. When it wants to use a message, it needs to retrieve the codeword symbols from storage servers, decode them, and then decrypt them by using cryptographic keys. 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. Finally, besides data storing and retrieving, it is hard for storage servers to directly support other functions. storage 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.

Related work

Distributed Storage Systems

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

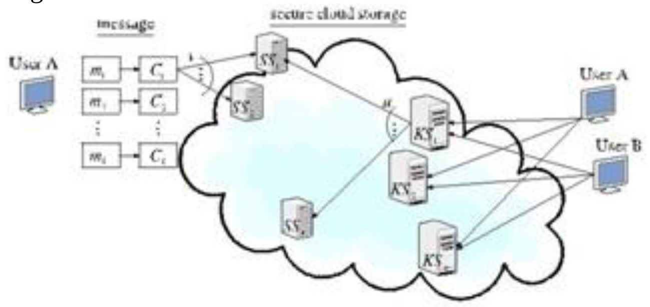


Figure 1. Distributed storage system

Proxy Re-Encryption Schemes

Proxy re-encryption schemes are proposed by Mambo and Okamoto [14] and Blaze et al. [15]. In a proxy re-encryption scheme, a proxy server can transfer a ciphertext under a public key PKA to a new one under another public key PKB by using the re-encryption key RKA!B. The server does not know the plaintext during transformation. Ateniese et al. [16] proposed some proxy re-encryption schemes and applied them to the

Tele:
E-mail addresses: manikkannan@gmail.com

© 2014 Elixir All rights reserved

sharing function of secure storage systems. In their work, messages are first encrypted by the owner and then stored in a storage server. When a user key server KSi holds a key share SKA;i, 1 _ i _ m. The key is shared with a threshold t.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. , 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 storage servers. Each storage 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 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 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, each key server KSi requests u randomly chosen storage servers to get codeword symbols and does partial decryption on the received codeword symbols by using the key share SKA;i. Finally, user A combines the partially decrypted codeword symbols to obtain the original message M.

Integrity Checking Functionality

The 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 [20], [21] and the notion of proof of storage [22], [23], [24] are proposed. Public auditability of stored data is addressed in [25]. Nevertheless all of them consider the messages in the cleartext form.

Methods and materials used

Rsa 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 message or data then only the person with the private key can read this message , not even the person who encoded the original message with the public key can now decoded this. The idea behind this to prevent" man in the middle attacks".

Public Key

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

Symmetric Vs Asymmetric Algorithm

When using symmetric algorithms, both parties share the same key for en-and decryption. To provide privacy, this key needs to be kept secret. Once somebody else gets to know the key, it is not safe anymore. Symmetric algorithm have 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 want to send encrypted message , therefore called " Public key". Everybody having the public key is able to send encrypted message to the owner of the secret key. The secret key can`t be reconstructed from the public key. Asymmetric algorithm are much slower than symmetric . Therefore, in many application, a combination 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.

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. Message 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.

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 is based on the user given inputs.

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 scheme used for providing high security. This is first step storing process.

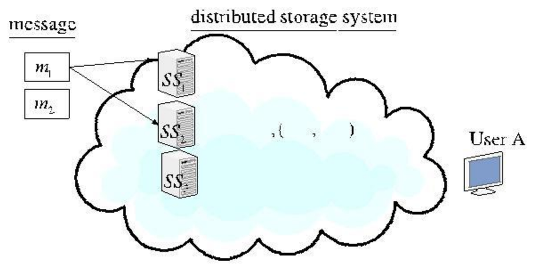


Figure 2 Encryption

Proxy Re-Encryption Scheme

Proxy re-encryption schemes 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. Cryptosystem consists of three algorithms. That is One for key generation, One for encryption, One for decryption. A proxy server is a server that acts as an intermediate for request from clients seeking resources from other servers. In the proposed system encrypted files are re-encrypted usinf threshold proxy re-encryption scheme.

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.

Data Retrieval On Cloud

Data are downloaded form 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.

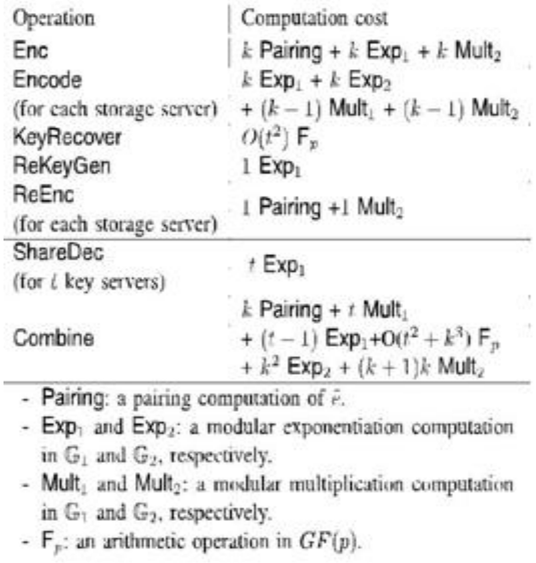
Results and discussion

A Secure Cloud Storage System with Secure Forwarding

Data retrieval. There are two cases for the data retrieval phase. The first case is that a user A retrieves his own message. When user A wants to retrieve the message with the identifier ID, he informs all key servers with the identity token _. A key server first retrieves original codeword symbols from u randomly chosen storage servers and then performs partial decryption ShareDecð_Þ on every retrieved original codeword symbol C0. The result of partial decryption is called a partially decrypted codeword symbol. The key server sends the partially decrypted codeword symbols _ 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 storage servers, he executes Combineð_Þ on the t partially decrypted codeword symbols to recover the blocks m1; m2; . . . ; mk. The second case is that a user B retrieves a message 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 symbols and perform partial decryption Share-Decð_Þ on re-encrypted codeword symbols.

Analysis

The computation cost by the number of pairing operations, modular exponentiations in G1 and G2, modular multiplications in G1 and G2, and arithmetic operations over GF ðpÞ. These operations 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 Mult$_2$. The methodology of analysis is similar to that in [13] and [6]. However, we consider a different system 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 system model and the one in [13] is that our system model has key servers. In [13], a single user queries k distinct storage servers to retrieve the data. On the other hand, each key server in our system independently queries u storage 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 $\hat{e}$.
- Exp$_1$ and Exp$_2$: a modular exponentiation computation in G$_1$ and G$_2$, respectively.
- Mult$_1$ and Mult$_2$: a modular multiplication computation in G$_1$ and G$_2$, respectively.
- F$_p$: an arithmetic operation in $GF(p)$.

Conclusions

The performance and security will be high compared to the existing system because of using encryption and re-encryption, for the data stored in the cloud. In existing 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 proposed system, data are stored in multiple 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 scheme.

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 Persis-tent 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 Fail-ures,” Proc. Second Symp. Networked Systems Design and Implemen-tation (NSDI), pp. 143-158, 2005.

[3] Z. Wilcox-O’Hearn and B. Warner, “Tahoe: The Least-Authority Filesystem,” 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.

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.

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

[8] 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.

[9] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran, “Ubiqui-tous 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.

[10] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran, “Decen-tralized Erasure Codes for Distributed Networked Storage,” IEEE Trans. Information Theory, vol. 52, no. 6 pp. 2809-2816, June 2006.

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

[12] M. Blaze, G. Bleumer, and M. Strauss, “Divertible Protocols and Atomic Proxy Cryptography,” Proc. Int’l Conf.

Theory and Applica-tion of Cryptographic Techniques (EUROCRYPT), pp. 127-144, 1998.
[13] 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.
[14] 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.
[15] G. Ateniese, K. Benson, and S. Hohenberger, “Key-Private Proxy Re-Encryption,” Proc. Topics in Cryptology (CT-RSA), pp. 279-294, 2009.
[16] 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.
[17] 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.
[18] 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 (SecureComm), pp. 1-10, 2008.
[19] 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.
[20] 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.
[21] A. Shamir, “How to Share a Secret,” ACM Comm., vol. 22, pp. 612-613, 1979.
[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 Deduplication Clusters,”