ISSN (ONLINE) : 2395-695X
ISSN (PRINT) : 2395-695X
Available online at www.ijarbest.com
International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST)
Vol. 1, Issue 9, December 2015

Information Sharing Using Homomorphic Re-crypt In Cloud Storage

B.Vijaya Nirmala[1], N.Deepa[2], V.R.Arulmozhi[3]
[1,2,3] Assistant Professor/Department of CSE, RVS Educational Trust's Group of Institution Dindigul,Tamilnadu, India.
[1]bvijayanirmalacse@gmail.com
[2]deepanatrayan@gmail.com
[3]arulmozhiram@gmail.com

***Abstract*— Cloud computing, empowers clients to remotely store their information in a cloud, in order to enjoy services on demand. With quick improvement of cloud computing, more ventures will outsource their delicate information for partaking in a cloud. To keep the common information classified against untrusted cloud service providers (CSPs), a characteristic path is to store just the scrambled information in a cloud. To take care of the issue ,a completely homomorphic encryption plan Scalable Data Sharing in Cloud Storage which has both generally little key and cipher text key size. Our development takes after delivering a completely homomorphic plan people in general and private keys comprise of two extensive whole numbers (one of which is shared by both the public and private key) and the figure content comprises of one substantial number. In that capacity, our plan has smaller message expansion and key size original scheme . Fully Homomorphic Public Key Encryption cipher text is an integer rather than a vector.expect that our plan is secure under key ward encryptions, absolutely to keep the documentation less complex, to manage the more broad case is prompt.**

Keywords—Cloud service providers; Ciphertext; Public-key; Homomorphic Encryption.

I. INTRODUCTION

Cloud storage is gaining popularity recently. In enterprise settings, we can see the rise in demand for data outsourcing, which assists in the strategic management of corporate data. It is also used as a core technology behind many online services for personal applications. Nowadays, it is easy to apply for free accounts for email, photo album, file sharing and/or remote access, with storage size more than 25 GB (or a few dollars for more than 1 TB). Together with the current wireless technology, users can access almost all of their files and emails by a mobile phone in any corner of the world.

Cloud computing, as an emerging computing paradigm, enables users to remotely store their data into a cloud so as to enjoy scalable services on-demand. Especially for small and medium-sized enterprises with limited budgets, they can achieve cost savings and productivity. However, allowing cloud service providers (CSPs), which are not in the same trusted domains as enterprise users, to take care of confidential data, may raise potential security and privacy issues. To keep the sensitive user data confidential against untrusted CSPs, a natural way is to apply cryptographic approaches, by disclosing decryption keys only to authorized users. However, when enterprise users outsource confidential data for sharing on cloud servers, the adopted encryption system should not only support fine-grained access control, but also provide high performance, full delegation, and scalability, so as to best serve the needs of accessing data anytime and anywhere.Efficient search on encrypted data is also an important concern in clouds. The clouds should not know the query but should be able to return the records that satisfy the query. This is achieved by means of searchable encryption [3], [4]. The keywords are sent to the cloud encrypted, and the cloud returns the result without knowing the actual keyword for the search. The problem here is that the data records should have keywords associated with them to enable the search. The correct records are returned only

When searched with the exact keywords. Several trends are opening up the era of Cloud Computing.The ever cheaper and more powerful processors, together with the "software as a service" (SaaS) computing architecture, are transforming data centres into pools of computing service on a huge scale. Meanwhile, the increasing network bandwidth and reliable yet flexible network connections make it even possible that clients can now subscribe high-quality services from data and software that reside solely on remote data centres.

Although envisioned as a promising service platform for the Internet, this new data storage paradigm in "Cloud" brings about many challenging design issues which have profound influence on the security and performance of the overall system. One of the biggest concerns with cloud data storage is that of data integrity verification at untrusted servers. What is more serious is that for saving money and storage space the service provider might neglect to keep or deliberately delete rarely accessed data files which belong to an ordinary client. Consider the large size of the outsourced electronic data and the client's constrained resource capability, the core of the problem can be generalized as how can the client find an efficient way to perform periodical integrity verifications without the local copy of data files.

1.1 Our Contributions

In modern cryptography, a fundamental problem we often study is about leveraging the secrecy of a small piece of knowledge into the ability to perform cryptographic functions (e.g., encryption, authentication) multiple times. In this paper, we study how to make a decryption key more powerful in the sense that it allows decryption of multiple ciphertexts, without increasing its size. Specifically, our problem statement is "To

ISSN (ONLINE) : 2395-695X
ISSN (PRINT) : 2395-695X
Available online at www.ijarbest.com
International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST)
Vol. 1, Issue 9, December 2015

design an efficient public-key encryption scheme which supports flexible delegation in the sense that any subset of the ciphertexts (produced by the encryption scheme) is decryptable by a constant-size decryption key (generated by the owner of the master-secret key).” We solve this problem by introducing a special type of public-key encryption which we call key-aggregate cryptosystem (KAC).

In KAC, users encrypt a message not only under a public-key, but also under an identifier of ciphertext called class. That means the ciphertexts are further categorized into different classes. The key owner holds a master-secret called master-secret key, which can be used to extract secret keys for different classes. More importantly, the extracted key have can be an aggregate key which is as compact as a secret key for a single class, but aggregates the power of many such keys, i.e., the decryption power for any subset of ciphertext classes. With our solution, Alice can simply send Bob a single aggregate key via a secure e-mail. Bob can download the encrypted photos from Alice’s Dropbox space and then use this aggregate key to decrypt these encrypted photos. The scenario is depicted in Fig. 1.

The sizes of ciphertext, public-key, master-secret key, and aggregate key in our KAC schemes are all of constant size. The public system parameter has size linear in the number of ciphertext classes, but only a small part of it is needed each time and it can be fetched on demand from large (but non-confidential) cloud storage. Previous results may achieve a similar property featuring a constant-size decryption key, but the classes need to conform to some predefined hierarchical relationship. Our work is flexible in the sense that this constraint is eliminated, that is, no special relation is required between the classes.

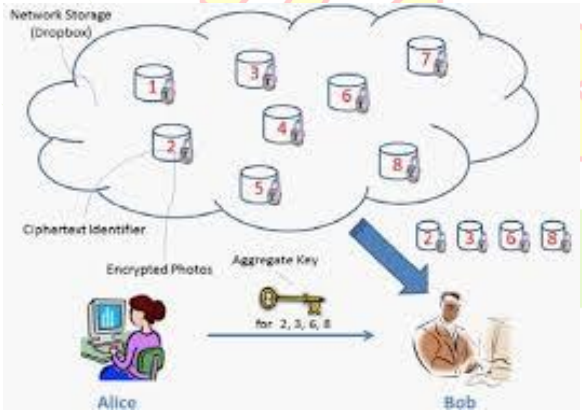


Fig. 1. Alice shares files with identifiers 2, 3, 6, and 8 with Bob by sending him a single aggregate key.

The detail and other related works can be found in Section 2. The existing system has several concrete KAC schemes with different security levels and extensions in this paper. All constructions can be proven secure in the standard model. To the best of our knowledge, our aggregation mechanism in KAC has not been investigated.

The detail and other related works can be found in Section 3. We propose the several concrete Fully

Homomorphic with different public key encryption for data sharing in cloud storage .

II. KEY-AGGREGATE ENCRYPTION

We first give the framework and definition for key aggregate encryption. Then we describe how to use KAC in a scenario of its application in cloud storage.

2.1 Framework

A key-aggregate encryption scheme consists of five polynomial-time algorithms as follows.

The data owner establishes the public system parameter via Setup and generates a public/master-secret key pair via KeyGen. Messages can be encrypted via Encrypt by anyone who also decides what ciphertext class is associated with the plaintext message to be encrypted. The data owner can use the master-secret to generate an aggregate decryption key for a set of ciphertext classes via Extract. The generated keys can be passed to delegates securely (via secure e-mails or secure devices) Finally, any user with an aggregate key can decrypt any ciphertext provided that the ciphertext’s class is contained in the aggregate key via Decrypt.

Setup : executed by the data owner to setup an account on an untrusted server. On input a security level parameter 1_ and the number of ciphertext classes n (i.e., class index should be an integer bounded by 1 and n), it outputs the public system parameter param, which is omitted from the input of the other algorithms for brevity.
KeyGen: executed by the data owner to randomly
generate a public/master-secret key pair (pk; msk).
Encrypt(pk, i, m): executed by anyone who wants to encrypt data. On input a public-key pk, an index I denoting the ciphertext class, and a message m, it outputs a ciphertext C.
Extract(msk, S): executed by the data owner for delegating the decrypting power for a certain set of ciphertext classes to a delegatee. On input the master-secret key msk and a set S of indices corresponding to different classes, it outputs the aggregate key for set S denoted by KS.
Decrypt(KS, S, i,C): executed by a delegatee who received an aggregate key KS generated by Extract. On input KS, the set S, an index i denoting the ciphertext class the ciphertext C belongs to, and C, it outputs the decrypted result m if i 2 S.

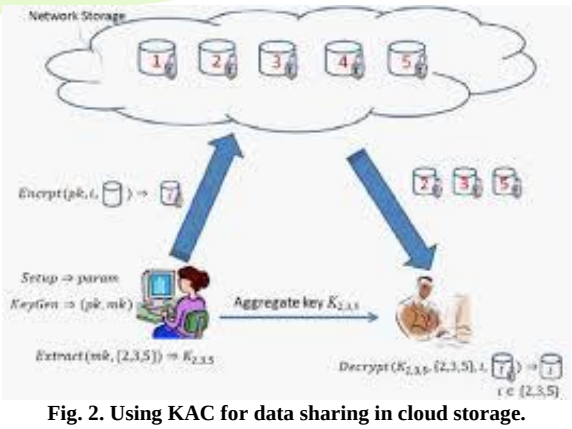


Fig. 2. Using KAC for data sharing in cloud storage.

ISSN (ONLINE) : 2395-695X
ISSN (PRINT) : 2395-695X
Available online at www.ijarbest.com
International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST)
Vol. 1, Issue 9, December 2015

2.2 Sharing Encrypted Data

A canonical application of KAC is data sharing. The key aggregation property is especially useful when we expect the delegation to be efficient and flexible. The schemes enable a content provider to share her data in a confidential and selective way, with a fixed and small ciphertext expansion, by distributing to each authorized user a single and small aggregate key.

III. FULLY HOMOMORPHIC PUBLIC KEY ENCRYPTION

Homomorphic scheme into a fully homomorphic scheme that our scheme is a specialization of Gentry's scheme, we only need to recast Gentry's method for our parameters. Indeed we can simplify the method somewhat, since our cipher text is an integer rather than a vector depicted in fig 3. Assume that our scheme is secure under key dependent encryptions, purely to keep the notation simpler; to deal with the more general case is immediate from our discussion. At a high level we need to define a new algorithm called Re crypt, which takes a cipher text c and re-encrypts it to $c$new, whilst at the same time removing some of the errors in c. Intuitively this takes a "dirty cipher text" c and "cleans it" to obtain the cipher text $c$new. The encryption key with some additional information, by extending the algorithm Key Gen with the following additional operations, based on two integer parameters $s1$ and $s2$. We make use of the fact that we are only interested in the coefficients of $Z(x)$ modulo $2p$.

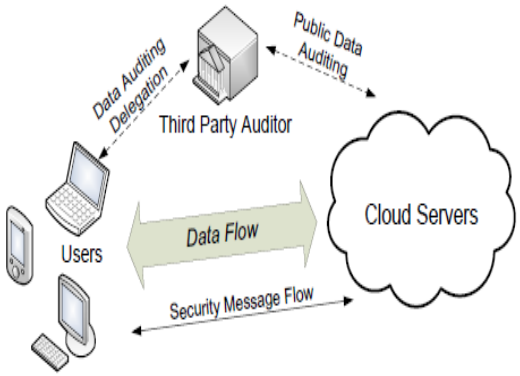


Fig 3. System Architecture Fully Homomorphic Encryption

Generate $s1$ uniformly random integers Bi in $[-p, \ldots, p]$ such that there exists a subset S of $s2$ elements

$$\sum_{j \in S} B_j = B$$

Define sk$i$ = 1 if $i \in S$ and 0 otherwise. Notice that only $s2$ of the bits {sk$i$} are set to one. Encrypt the bits sk$i$ under the somewhat homomorphic scheme to obtain ci ← Encrypt(sk$i$, PK).The public key now consists of PK = (p, α, $s1$, $s2$, {ci,Bi}$s1$ i=1) .

The re-encryption operation Re crypt(c, PK): This algorithm takes as input a "dirty" cipher text c, and then produces a "cleaner" cipher text $c$new of the same message,

but with less "errors" in its randomization vector. The re-encryption works by performing a homomorphic decryption on an encryption of the cipher texts bits. The system parameter can also be generated by a trusted party, shared between all users and even hard-coded to the user program (and can be updated via "patches").

IV. PERFORMANCE ANALYSIS

4.1 Compression Factors

For a concrete comparison, we investigate the space requirements of the tree-based key assignment approach we described in Section 2.1. This is used in the complete subtree scheme, which is a representative solution to the broadcast encryption problem following the well-known subset-cover framework [33]. It employs a static logical key hierarchy, which is materialized with a full binary key tree of height h (equals to 3 in Fig. 2), and thus can support up to 2^h ciphertext classes, a selected part of which is intended for an authorized delegatee. In an ideal case as depicted in Fig. 3a, the delegatee can be granted the access to 2^{hs} classes with the possession of only one key, where hs is the height of a certain subtree (e.g., hs ¼ 2).

On the other hand, to decrypt ciphertexts of a set of classes, sometimes the delegatee may have to hold a large number of keys is depicted in 4a . Therefore, we are interested in na, the number of symmetric keys to be assigned in this hierarchical key approach, in an average sense. We assume that there are exactly 2h ciphertext classes, and the delegatee of concern is entitled to a portion r of them. That is, r is the delegation ratio, the ratio of the delegated ciphertext classes to the total classes. A comparison of the number of granted keys between three methods is depicted in Fig. 4b. We can see that if we grant the key one by one, the number of granted keys would be equal to the number of the delegated ciphertext classes. With the tree-based structure, we can save a number of granted keys according to the delegation ratio. On the contrary, in our proposed approach, the delegation of decryption can be efficiently implemented with the aggregate key, which is only of fixed size.

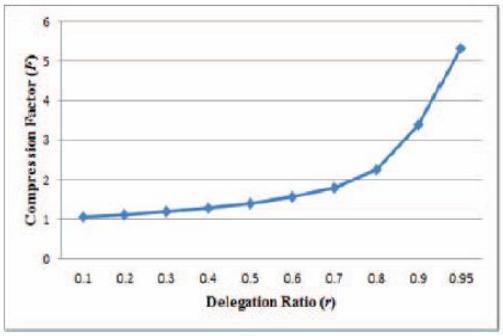
Obviously, if r ¼ 0, na should also be 0, which means no access to any of the classes, if r ¼ 100%, na should be as low as 1, which means that the possession of only the root key in the hierarchy can grant the access to all the 2h classes. Consequently, one may expect that na may first increase with r, and may decrease later. We set r ¼ 10%; 20%; . . . ; 90%, and choose the portion in a random manner to model an arbitrary "delegation pattern"

In our experiment, the delegation is randomly chosen. It models the situation that the needs for delegating to different users may not be predictable as time goes by, even after a careful initial planning. This gives empirical evidences to support our thesis that hierarchical key assignment does not save much in all cases.

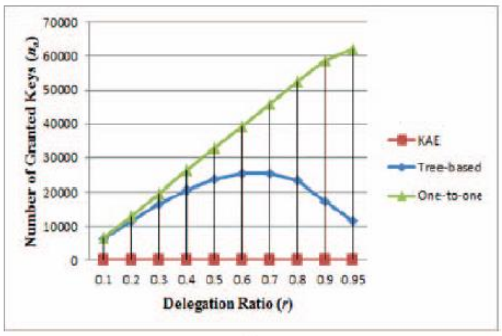
ISSN (ONLINE) : 2395-695X
ISSN (PRINT) : 2395-695X
Available online at www.ijarbest.com
International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST)
Vol. 1, Issue 9, December 2015

4.2 Performance of Our Proposed Schemes

Our approaches allow the compression factor F (F ¼ n in our schemes) to be a tunable parameter, at the cost of O(n)-sized system parameter. Encryption can be done in constant time, while decryption can be done in O(Sj) group multiplications (or point addition on elliptic curves) with two pairing operations, where S is the set of ciphertext classes decryptable by the granted aggregate key and jSj _ n. As expected, key extraction requires O(Sj) group multiplications as well, which seems unavoidable. However, as demonstrated by the experiment results, we do not need to set a very high n to have better compression than the tree-based approach. Note that group multiplication is a very fast operation. Again, we confirm empirically that our analysis is true.



(a)



(b)

Fig. 4. (a) Compression achieved by the tree-based approach for delegating different ratio of the classes. (b) Number of granted keys (na) required for different approaches in the case of 65,536 classes of data.

The basic Fully Homomorphic encryption system is implemented in C with the pairing-based cryptography (PBC) Library8 version 0.4.18 for the underlying elliptic-curve group and pairing operations. Since the granted key can be as small as one GG element, and the ciphertext only contains two GG and one GGT elements, we used (symmetric) pairings over Type-A (supersingular) curves as defined in the PBC library which offers the highest efficiency among all types of curves, even though Type-A curves do not provide the shortest representation for group elements. In our implementation, p is a 160-bit Solinas prime, which offers 1,024-bit of discrete-logarithm security. With this Type-A curves setting in PBC.

TABLE 1
Performance of Our Basic Construction for h ¼ 16 with Respect to Different Delegation Ratio r (in Milliseconds)

r	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95
Setup	8.4									
Extract	2	4	5	7	8	9	10	10	11	11
Decrypt	4	6	9	12	14	15	16	18	20	20

Our experiment results are shown in Table 1. The execution times of Setup, KeyGen, and Encrypt are independent of the delegation ratio r. In our experiments, KeyGen takes 3.3 milliseconds and Encrypt takes 6.8 milliseconds. As expected, the running time complexities of Extract and Decrypt increase linearly with the delegation ratio r (which determines the size of the delegated set S). Our timing results also conform to what can be seen from the equation in Extract and Decrypt—two pairing operations take negligible time, the running time of Decrypt is roughly a double of Extract. Note that our experiments dealt with up to 65,536 number of classes (which is also the compression factor), and should be large enough for fine grained data sharing in most situations.

Finally, we remark that for applications where the number of cipher text classes is large but the non confidential storage is limited, one should deploy our schemes using the Type-D pairing bundled with the PBC. The system parameter requires approximately 2.6 megabytes, which is as large as a lower quality MP3 file or a higher resolution JPEG file that a typical cellphone can store more than a dozen of them. But we saved expensive secure storage without the hassle of managing a hierarchy of delegation classes.

V. CONCLUSION AND FUTURE WORK

It is been considered how to “compress” secret keys in public-key cryptosystems which support delegation of secret keys for different ciphertext classes in cloud storage. No matter which one among the power set of classes, the delegate can always get an aggregate key of constant size. Our approach is more flexible than hierarchical key assignment which can only save spaces if all key-holders share a similar set of privileges. A limitation in our work is the predefined bound of the number of maximum ciphertext classes. In cloud storage, the number of ciphertexts usually grows rapidly. So we have to reserve enough ciphertext classes for the future extension. Otherwise, we need to expand the public-key. Although the parameter can be downloaded with ciphertexts, it would be better if its size is independent of the maximum number of ciphertext classes. On the other hand, when one carries the delegated keys around in a mobile device without using special trusted hardware, the key is prompt to leakage, designing a leakage-resilient cryptosystem, yet allows efficient and flexible key delegation is also an interesting direction.

ISSN (ONLINE) : 2395-695X
ISSN (PRINT) : 2395-695X
Available online at www.ijarbest.com
International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST)
Vol. 1, Issue 9, December 2015

REFERENCES

1. S.S.M. Chow, Y.J. He, L.C.K. Hui, and S.-M. Yiu, "SPICE – Simple Privacy-Preserving Identity-Management for Cloud Environment," Proc. 10th Int'l Conf. Applied Cryptography and Network Security (ACNS), vol. 7341, pp. 526-543, 2012.
2. C. Wang, S.S.M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Secure Cloud Storage," IEEE Trans. Computers, vol. 62, no. 2, pp. 362-375, Feb. 2013.
3. B. Wang, S.S.M. Chow, M. Li, and H. Li, "Storing Shared Data on the Cloud via Security-Mediator," Proc. IEEE 33rd Int'l Conf. Distributed Computing Systems (ICDCS), 2013.
4. S.S.M. Chow, C.-K. Chu, X. Huang, J. Zhou, and R.H. Deng, "Dynamic Secure Cloud Storage with Provenance," Cryptography and Security, pp. 442-464, Springer, 2012.
5. G. Ateniese, A.D. Santis, A.L. Ferrara, and B. Masucci, "Provably-Secure Time-Bound Hierarchical Key Assignment Schemes," J. Cryptology, vol. 25, no. 2, pp. 243-270, 2012.
6. S.S.M. Chow, Y. Dodis, Y. Rouselakis, and B. Waters, "Practical Leakage-Resilient Identity-Based Encryption from Simple Assumptions," Proc. ACM Conf. Computer and Comm. Security, pp. 152-161, 2010.
7. T. Okamoto and K. Takashima, "Achieving Short Ciphertexts or Short Secret-Keys for Adaptively Secure General Inner-Product Encryption," Proc. 10th Int'l Conf. Cryptology and Network Security (CANS '11), pp. 138-159, 2011.
8. S.S.M. Chow, J. Weng, Y. Yang, and R.H. Deng, "Efficient Unidirectional Proxy Re-Encryption," Proc. Progress in Cryptology (AFRICACRYPT '10), vol. 6055, pp. 316-332, 2010.
9. T.H. Yuen, S.S.M. Chow, Y. Zhang, and S.M. Yiu, "Identity-Based Encryption Resilient to Continual Auxiliary Leakage," Proc. Advances in Cryptology Conf. (EUROCRYPT '12), vol. 7237, pp. 117-134, 2012.
10. M. Chase and S.S.M. Chow, "Improving Privacy and Security in Multi-Authority Attribute-Based Encryption," Proc. ACM Conf. Computer and Comm. Security, pp. 121-130. 2009.
11. J. Benaloh, "Key Compression and Its Application to Digital Fingerprinting," technical report, Microsoft Research, 2009.
12. B. Alomair and R. Poovendran, "Information Theoretically Secure Encryption with Almost Free Authentication," J. Universal Computer Science, vol. 15, no. 15, pp. 2937-2956, 2009.
13. M.J. Atallah, M. Blanton, N. Fazio, and K.B. Frikken, "Dynamic and Efficient Key Management for Access Hierarchies," ACM Trans. Information and System Security, vol. 12, no. 3, pp. 18:1-18:43, 2009.
14. J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, "Patient Controlled Encryption: Ensuring Privacy of Electronic Medical Records," Proc. ACM Workshop Cloud Computing Security (CCSW '09), pp. 103-114, 2009.
15. L. Hardesty, Secure Computers Aren't so Secure. MIT press, http://www.physorg.com/news176107396.html, 2009.