International Journal of Engineering Research and General Science Volume 3, Issue 2, Part 2, March-April, 2015
ISSN 2091-2730

Key Aggregate Cryptosystem with Identity Based Encryption for Data Sharing in Cloud Storage

Ms. Farog Fatema Khan[1], Ms. Kshitija Mohod[2], Prof. V.B. Gadicha[3]

[1] B.E. (C.S.E, Final Year), P.R. Patil College of Engineering and Technology, Amravati, India
Email: faroghfatema77@gmail.com, Contact No.: 8928241808
[2]B.E. (C.S.E, Final Year), P.R. Patil College of Engineering and Technology, Amravati, India
[3]H.O.D. (C.S.E), P.R. Patil College of Engineering and Technology, Amravati, India
Email: v_gadicha@rediffmail.com, Contact No.: 9423622833

Abstract—
Using Cloud Storage, users can remotely store and can share their data and enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources, without the burden of local data storage and maintenance. Data sharing is one of the important functionality in cloud storage. We show how to securely, efficiently, and flexibly share data with others in cloud storage. We describe new public-key cryptosystems which produce constant-size ciphertexts where one can aggregate any set of secret keys and make them as compact as a single key, but encompassing the power of all the keys being aggregated. In other words, the secret key holder can release a constant-size aggregate key for flexible choices of ciphertext set in cloud storage, but the other encrypted files outside the set remain confidential. This compact aggregate key can be conveniently sent to others or be stored in a smart card with very limited secure storage.

Keywords: Cloud Storage, Data Sharing, Key-Aggregate Encryption, Public Key Cryptosystem, Cloud Computing, Public-Key Encryption, Confidentiality.

I. INTRODUCTION

Cloud computing is a model for convenient and on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management efforts. Main goal of the cloud computing is to provide scalable and inexpensive on-demand computing infrastructures with good quality of service levels. Many developers of cloud-based applications struggle to include security. In other cases, developers simply cannot provide real security with currently affordable technological capabilities. The architecture of the Cloud Computing involves multiple cloud components interacting with each other about the various data they are holding on too, thus helping the user to get to the required data on a faster rate.

Cloud systems can be used to enable data sharing capabilities and this can provide an abundant of benefits to the user. There is currently a push for IT organizations to increase their data sharing efforts. The benefits organizations can gain from data sharing is higher productivity. With multiple users from different organizations contributing to data in the Cloud, the time and cost will be much less compared to having to manually exchange data and hence creating a clutter of redundant and possibly out-of-date documents.

Data sharing is an important functionality in cloud storage. The challenging problem is how to effectively share encrypted data. Users should be able to delegate the access rights of the sharing data to others so that they can access these data from the server directly. However, finding an efficient and secure way to share partial data in cloud storage is not trivial. Below we will take Dropbox as an example for illustration [1].

Assume that Alice puts all her private photos on Dropbox, and she does not want to expose her photos to everyone. Due to various data leakage possibilities, so she encrypts all the photos using her own keys before uploading. One day, Alice's friend, Bob, asks her to share the photos taken over all these years which Bob appeared in. Alice can then use the share function of Dropbox, but the problem now is how to delegate the decryption rights for these photos to Bob [1].

We solve this problem by introducing a special type of public-key encryption which we call key-aggregate cryptosystem. In, key-aggregate cryptosystem 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 [1].

With 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 Figure 1.The sizes of ciphertext, public-key, and master-secret key and aggregate key in our key-aggregate cryptosystem schemes are all of constant size [1].

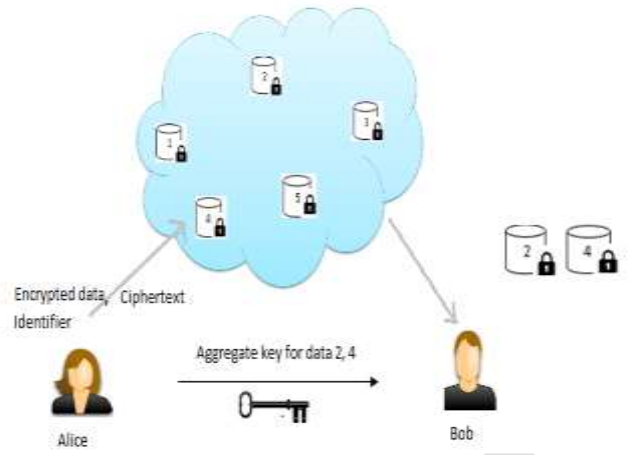

Fig: 1. Alice shares files with identifiers 2, 4 with Bob by sending a single aggregate key.

II. RELATED WORK:

This section gives a brief introduction into the related work done on this subject.

A. CRYPTOGRAPHIC KEYS FOR A PREDEFINED HIERARCHY

We start by discussing the most relevant study in the literature of cryptography/security. Cryptographic key assignment schemes (e.g., [2], [3], [4], [5]) aim to minimize the expense in storing and managing secret keys for general cryptographic use. Utilizing a tree structure, a key for a given branch can be used to derive the keys of its descendant nodes (but not the other way round). Just granting the parent key implicitly grants all the keys of its descendant nodes. Sandhu [6] proposed a method to generate a tree hierarchy of symmetrickeys by using repeated evaluations of pseudorandom function/block-cipher on a fixed secret. The concept can be generalized from a tree to a graph. More advanced cryptographic key assignment schemes support access policy that can be modeled by an acyclic graph or a cyclic graph [7], [8], [9].

Most of these schemes produce keys for symmetric-key cryptosystems, even though the key derivations may require modular arithmetic as used in public-key cryptosystems, which are generally more expensive than "symmetric-key operations" such as pseudorandom function. We take the tree structure as an example. Alice can first classify the ciphertext classes according to their subjects like Figure 2(a). Each node in the tree represents a secret key, while the leaf node represents the keys for individual ciphertext classes. Filled circles represent the keys for the classes to be delegated and circles circumvented by dotted lines represent the keys to be granted.

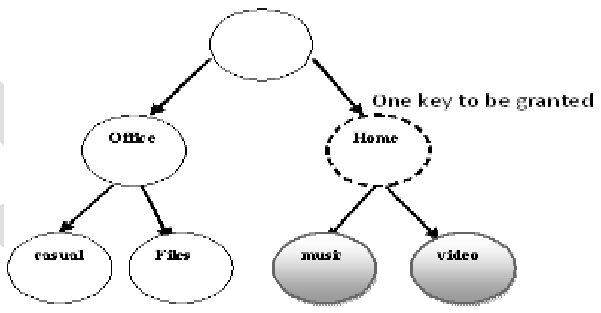

Fig: 2(a). Compact key is not always possible for a fixed hierarchy

Note that every key of the non-leaf node can derive the keys of its descendant nodes. In Figure 2(a), if Alice wants to share all the files in the "home" category, she only needs to grant the key for the node "home", which automatically grants the delegatee the keys of all the descendant nodes ("video", "music"). This is the ideal case, where most classes to be shared belong to the same branch and thus a parent key of them is sufficient. However, it is still difficult for general cases.

As shown in Figure 2(b), if Alice shares her demo music at work ("office"! "Casual" and "office"! "files") with a colleague who also has the rights to see some of her personal data, what she can do is to give more keys, which leads to an increase in the total key size. For this delegatee in our example, the number of granted secret keys becomes the same as the number of classes.

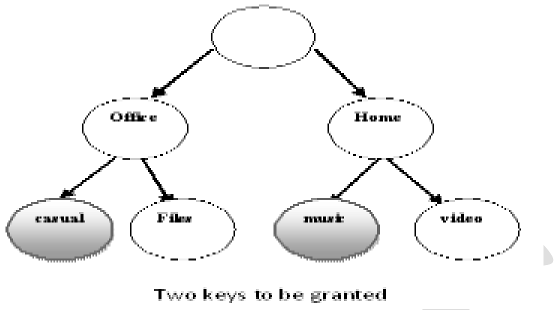

Fig: 2(b). Compact key is not always possible for a fixed hierarchy

In general, hierarchical approaches can solve the problem partially if one intends to share all files under a certain branch in the hierarchy. On average, the number of keys increases with the number of branches. It is unlikely to come up with a hierarchy that can save the number of total keys to be granted for all individuals (which can access a different set of leaf-nodes) simultaneously.

B. COMPACT KEY IN IDENTITY-BASED ENCRYPTION

Identity-based encryption (IBE) (e.g., [10], [11], [12]) is a type of public-key encryption in which the public-key of a user can be set as an identity-string of the user (e.g., an email address). There is a trusted party called private key generator (PKG) in IBE which holds a master-secret key and issues a secret key to each user with respect to the user identity. The encryptor can take the public parameter and a user identity to encrypt a message. The recipient can decrypt this ciphertext by his secret key. Guo et al. [13], [14] tried to build IBE with key aggregation. One of their schemes [13] assumes random oracles but another [14] does not.

In their schemes, key aggregation is constrained in the sense that all keys to be aggregated must come from different "identity divisions". While there are an exponential number of identities and thus secret keys, only a polynomial number of them can be aggregated. Most importantly, their key-aggregation [13], [14] comes at the expense of *O(n)* sizes for both ciphertexts and the public parameter, where n is the number of secret keys which can be aggregated into a constant size one. This greatly increases the costs of storing and transmitting ciphertexts, which is impractical in many situations such as shared cloud storage. As we mentioned, our schemes feature constant ciphertext size, and their security holds in the standard model. In fuzzy IBE [11], one single compact secret key can decrypt ciphertexts encrypted under many identities which are close in a certain metric space, but not for an arbitrary set of identities and therefore it does not match with our idea of key aggregation.

III. KEY-AGGREGATE ENCRYPTION

Here we are describing how to use key-aggregate cryptosystem in a scenario of its application in cloud storage. 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 delegatees 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* $(1^{\lambda}, n)$: The data owner establishes system parameter via *Setup*. On input a security level parameter 1^{λ} and the number of ciphertext classes n, it outputs the public system parameter *param*.
- *KeyGen:* It is executed by the data owner to randomly generate a public/master-secret key pair (*pk*, *msk*).
- *Encrypt (pk, i, m):* It is 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):* It is executed by the data owner for delegating the decrypting power for a certain set of ciphertext classes to a delegatee. On input the mastersecret key *msk* and a set *S* of indices corresponding to different classes, it outputs the aggregate key for set *S* denoted by K_s.
- *Decrypt* (K_s, S, i, C): It is executed by a delegatee who received an aggregate key K_s generated by *Extract.* On input K_s, the set *S*, an index *i* denoting the ciphertext class the ciphertext C belongs to, and C, it outputs the decrypted result *m* if $i \in S$.

A canonical application of key-aggregate cryptosystem 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. Here we describe the main idea of data sharing in cloud storage using key-aggregate cryptosystem, illustrated in Figure 3.

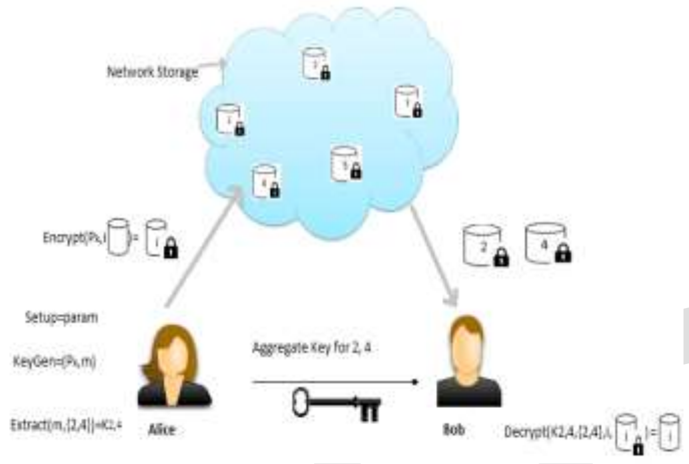

Fig: 3. Using key-aggregate cryptosystem for data sharing in cloud storage

Suppose Alice wants to share her data m_1, m_2,. . .$m_{i,}$ on the server. She first performs *Setup* $(1^{\lambda}, n)$ to get *param* and execute *KeyGen* to get the public/master-secret key pair *(pk, msk)*. The system parameter *param* and public-key *pk* can be made public and master-secret key *msk* should be kept secret by Alice. Anyone (including Alice herself) can then encrypt each m_i by C_i = *Encrypt (pk, i, m*). The encrypted data are uploaded to the server. With *param* and *pk*, people who cooperate with Alice can update Alice's data on the server. Once Alice is willing to share a set S of her data with a friend Bob, she can compute the aggregate key K_s for Bob by performing *Extract (msk, S).* Since K_s is just a constant size key, it is easy to be sent to Bob via a secure e-mail. After obtaining the aggregate key, Bob can download the data he is authorized to access. That is, for each $i \epsilon S$, Bob downloads C_i (and some needed values in *param*) from the server. With the aggregate key K_s, Bob can decrypt each C_i by *Decrypt* (K_s, *S, I,* C_i) for each $i \epsilon S$.

IV. CONCLUSION AND FUTURE WORK

How to protect users' data privacy is a central question of cloud storage. With more mathematical tools, cryptographic schemes are getting more versatile and often involve multiple keys for a single application. In this article, we consider 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 delegatee 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 the 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 [12], [21] yet allows efficient and flexible key delegation is also an interesting direction.

REFERENCES:

[1] Cheng-Kang Chu ,Chow, S.S.M, Wen-GueyTzeng, Jianying Zhou, and Robert H. Deng , "Key-Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage", IEEE Transactions on Parallel and Distributed Systems. Volume: 25, Issue: 2. Year :2014.
[2] S. G. Akl and P. D. Taylor, "Cryptographic Solution to a Problem of Access Control in a Hierarchy," ACM Transactions on Computer Systems (TOCS), vol. 1, no. 3, pp. 239–248, 1983.
[3] G. C. Chick and S. E. Tavares, "Flexible Access Control withMaster Keys," in Proceedings of Advances in Cryptology – CRYPTO '89, ser. LNCS, vol. 435. Springer, 1989, pp. 316–322.
[4] W.-G. Tzeng, "A Time-Bound Cryptographic Key Assignment Scheme for Access Control in a Hierarchy," IEEE Transactions onKnowledge and Data Engineering (TKDE), vol. 14, no. 1, pp. 182–188,2002.

[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] R. S. Sandhu, “Cryptographic Implementation of a Tree Hierarchy for Access Control,” Information Processing Letters, vol. 27, no. 2, pp. 95–98, 1988.
[7] Y. Sun and K. J. R. Liu, “Scalable Hierarchical Access Control inSecure Group Communications,” in Proceedings of the 23th IEEEInternational Conference on Computer Communications (INFOCOM ’04)IEEE, 2004.
[8] Q. Zhang and Y. Wang, “A Centralized Key Management Scheme for Hierarchical Access Control,” in Proceedings of IEEE Global Telecommunications Conference (GLOBECOM ’04). IEEE, 2004, pp. 2067–2071.
[9] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken, “Dynamic and Efficient Key Management for Access Hierarchies,” ACM Transactions on Information and System Security (TISSEC), vol. 12, no. 3, 2009.
[10] D. Boneh and M. K. Franklin, “Identity-Based Encryption from the Weil Pairing,” in Proceedings of Advances in Cryptology – CRYPTO ’01, ser. LNCS, vol. 2139. Springer, 2001, pp. 213–229.
[11] A. Sahai and B. Waters, “Fuzzy Identity-Based Encryption,” inProceedings of Advances in Cryptology - EUROCRYPT ’05, ser. LNCS, vol. 3494. Springer, 2005, pp. 457–473.
[12] S. S. M. Chow, Y. Dodis, Y. Rouselakis, and B. Waters, “Practical Leakage-Resilient Identity-Based Encryption from Simple Assumptions,” in ACM Conference on Computer and Communications Security, 2010, pp. 152–161.
[13]F. Guo, Y. Mu, and Z. Chen, “Identity-Based Encryption: How toDecrypt Multiple Ciphertexts Using a Single Decryption Key,” inProceedings of Pairing-Based Cryptography (Pairing ’07), ser. LNCS, vol. 4575. Springer, 2007, pp. 392–406.
[14] F. Guo, Y. Mu, Z. Chen, and L. Xu, “Multi-Identity Single-KeyDecryption without Random Oracles,” in Proceedings of InformationSecurity and Cryptology (Inscrypt ’07), ser. LNCS, vol. 4990.Springer, 2007, pp. 384–398.
[15] T. H. Yuen, S. S. M. Chow, Y. Zhang, and S. M. Yiu, “Identity Based Encryption Resilient to Continual Auxiliary Leakage,” in Proceedings of Advances in Cryptology - EUROCRYPT ’12, ser. LNCS, vol. 7237, 2012, pp. 117–134