International Journal of Science Engineering and Advance Technology

A Novel System for Scalable Data Sharing in Cloud Storage using Key-Aggregate

Nadendla Siva Rama Krishna[1], Y V V N Vara Prasd[2]
[1] M.Tech (CSE), Usha Rama College of Engineering and Technology, A.P., India.
[2]Associate Professor, Dept. of Information Technology, Usha Rama College of Engineering and Technology, A.P., India.

***Abstract* — Of late, rapid growth observed in the cloud technology. Data sharing is an essential functionality in cloud storage. In this paper, we show how to efficiently, securely and flexibly share data with others in cloud storage. We describe new public-key cryptosystems that produce constant-size ciphertexts such that efficient delegation of decryption rights for any set of ciphertexts is possible. The novelty is that 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. We provide formal security analysis of our schemes in the standard model. We also describe other application of our schemes. In particular, our schemes give the first public-key patient-controlled encryption for flexible hierarchy, which was yet to be known.**

***Keywords* — Cloud storage, data sharing, key-aggregate encryption, patient-controlled encryption**

I. INTRODUCTION

Storing data in a cloud is become a common phenomenon recently. In enterprise settings, we 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, and 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.

Considering data privacy, a traditional way to ensure it is to rely on the server to enforce the access control after authentication (e.g., [1]), which means any unexpected privilege escalation will expose all data. In a shared-tenancy cloud computing environment, things become even worse. Data from different clients can be hosted on separate virtual machines (VMs) but reside on a single physical machine. Data in a target VM could be stolen by instantiating another VM co-resident with the target one [2]. Regarding availability of files, there are a series of cryptographic schemes which go as far as allowing a third-party auditor to check the availability of files on behalf of the data owner without leaking anything about the data [3], or without compromising the data owners anonymity [4]. Likewise, cloud users probably will not hold the strong belief that the cloud server is doing a good job in terms of confidentiality. A cryptographic solution, for example, [5], with proven security relied on number-theoretic assumptions is more desirable, whenever the user is not perfectly happy with trusting the security of the VM or the honesty of the technical staff. These users are motivated to encrypt their data with their own keys before uploading them to the server.

Data sharing is an important functionality in cloud storage. For example, bloggers can let their friends view a subset of their private pictures; an enterprise may grant her employees access to a portion of sensitive data. The challenging problem is how to effectively share encrypted data. Of course users can download the encrypted data from the storage, decrypt them, then send them to others for sharing, but it loses the value of cloud storage. 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 Dropbox1 as an example for illustration.

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 possibility Alice cannot feel relieved by just relying on the privacy protection mechanisms provided by Dropbox, 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. A possible option Alice can choose is to securely send Bob the secret keys involved. Naturally, there are two extreme ways for her under the traditional encryption paradigm:

- Alice encrypts all files with a single encryption key and gives Bob the corresponding secret key directly.
- Alice encrypts files with distinct keys and sends Bob the corresponding secret keys.

Obviously, the first method is inadequate since all un-chosen data may be also leaked to Bob. For the second method, there are practical concerns on efficiency. The number of such keys is as many as the number of the shared photos, say, a thousand. Transferring these secret keys inherently requires a secure channel, and storing these keys requires rather expensive secure storage. The costs and complexities involved generally increase with the number of the decryption keys to be shared. In short, it is very heavy and costly to do that.

Encryption keys also come with two flavors—symmetric key or asymmetric (public) key. Using symmetric encryption, when Alice wants the data to be originated from a third party, she has to give the encryptor her secret key; obviously, this is not always desirable. By contrast, the encryption key and decryption key are different in publickey encryption. The use of public-key encryption gives more flexibility for our applications. For example, in enterprise settings, every employee can upload encrypted data on the cloud storage server without the knowledge of the company's master-secret key.

Therefore, the best solution for the above problem is that Alice encrypts files with distinct public-keys, but only sends Bob a single (constant-size) decryption key. Since the decryption key should be sent via a secure channel and kept secret, small key size is always desirable. For example, we cannot expect large storage for decryption keys in the resource-constraint devices like smart phones, smart cards, or wireless sensor nodes. Especially, these secret keys are usually stored in the tamper-proof memory, which is relatively expensive. The present research efforts mainly focus on minimizing the communication requirements (such as bandwidth, rounds of communication) like aggregate signature [6]. However, not much has been done about the key itself.

II. Our Contribution

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 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.

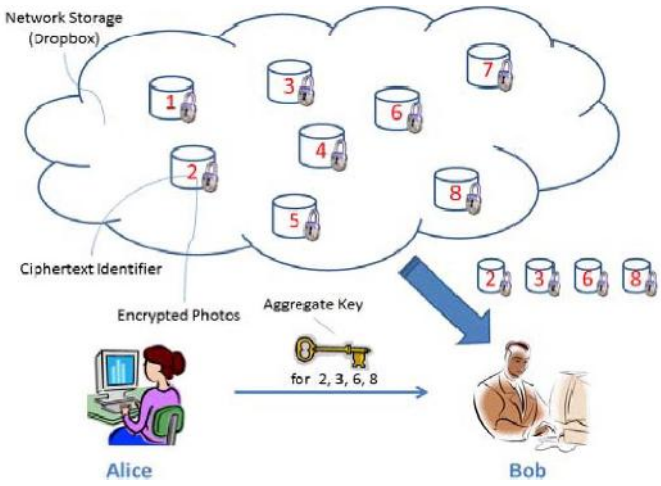

Figure 1 Alice share files with identifiers 2, 3, 6 and 8 with Bob by sending him a single aggregate key.

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.

We propose 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.

III. KEY-AGGREGATE ENCRYPTION

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(1^λ, n): executed by the data owner to setup an account on an untrusted server. On input a security level parameter 1 λ 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 randomlygenerate 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(K_S, 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 € S.

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.

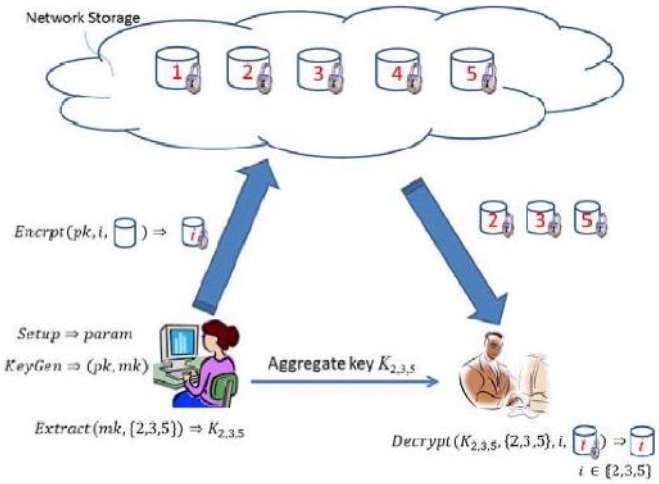

Figure 2 Using KAC for data sharing in cloud storage.

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.

Here, we describe the main idea of data sharing in cloud storage using KAC, illustrated in Fig. 2. Suppose Alice wants to share her data $m_1, m_2, \ldots, m_v$ on the server. She first performs Setup(1^λ, 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_i). 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 KS for Bob by performing Extract(msk, S). Since KS 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 € S, Bob downloads Ci (and some needed values in param) from the server. With the aggregate key KS, Bob can decrypt each Ci by Decrypt(K_S, S, ,C_i) for each i € S.

IV. Related Work

We start by discussing the most relevant study in the literature of cryptography/security. Cryptographic key assignment schemes ([8]) 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 proposed a method to generate a tree hierarchy of symmetric-keys by using repeated evaluations of pseudorandom function/blockcipher 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]. 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. Each node in the tree represents a secret key, while the leaf nodes 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. Note that every key of the non-leaf node can derive the keys of its descendant nodes.

However, it is still difficult for general cases. As shown in, if Alice shares her demo music at work ("work"! "casual"! "demo" and "work"! "confidential" ! "demo") 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. One can see that this approach is not flexible when the classifications are more complex and she wants to share different sets of files to different people. For this delegatee in our example, the number of granted secret keys becomes the same as the number of classes. 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.

V. Conclusion

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 paper, 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.

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 [9], [10] yet allows efficient and flexible key delegation is also an interesting direction.

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] L. Hardesty, Secure Computers Aren't so Secure. MIT press, http:// www.physorg.com/news176107396.html, 2009.

[3] 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.

[4] 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.

[5] 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.

[6] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," Proc. 22nd Int'l Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT '03), pp. 416-432, 2003.

[7] 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.

[8] 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.

[9] F. Guo, Y. Mu, Z. Chen, and L. Xu, "Multi-Identity Single-Key Decryption without Random Oracles," Proc. Information Security and Cryptology (Inscrypt '07), vol. 4990, pp. 384-398, 2007.

[10] 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 '06), pp. 89-98, 2006.