ISSN 2320 – 2602
Volume 4, No.3, March 2015
International Journal of Advances in Computer Science and Technology
Available Online at http://warse.org/pdfs/2015/ijacst03422015.pdf

KEY-AGGREGATE SECURE DATA SHARING IN CLOUD COMPUTING FOR CRYPTOSYSTEM

Ashraf Sabri Waheed Alameri, Y.V.K. Sudhakar

M.Sc (Computer Science) Mahatma Gandhi College Affiliated to Acharya Nagarjuna University Guntur, AP, India,ashfat2004@yahoo.com
M.Tech (CSE) Mahatma Gandhi College Affiliated to Acharya Nagarjuna University Guntur, AP, India, sudakartechcs@yahoo.com

ABSTRACT

Data sharing is an important functionality in cloud storage. In this paper, we show how to securely, efficiently, and flexibly share data with others in cloud storage. We describe new public-key cryptosystems that produce constant-size cipher texts such that efficient delegation of decryption rights for any set of cipher texts are 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 cipher text 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.

Key words: *Cloud storage, data sharing, key-aggregate encryption, patient-controlled encryption*

1. INTRODUCTION

LOUD storage is gaining popularity 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, 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 coresident 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 Drop box, 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 unchosen 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 public key 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 (see Section 3 for more details).

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 design an efficient public-key encryption scheme which supports flexible delegation in the sense that any subset of the cipher texts (produced by the encryption scheme) is decrypt able 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 cipher text called class.

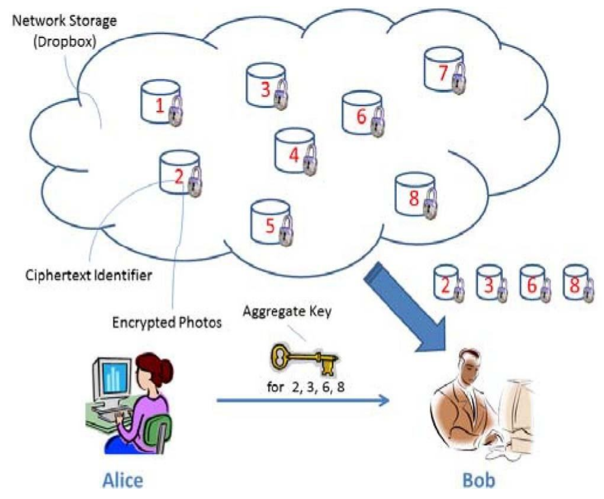

Figure. 1. Alice shares files with identifiers 2, 3, 6, and 8 with Bob by sending him a single aggregate key.

That means the cipher texts 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 cipher text 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 Drop box 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, 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 cipher text classes, but only a small part of it is

needed each time and it can be fetched on demand from large (but no 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. The detail and other related works can be found in Section 3. 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 mechanism2 in KAC has not been investigated.

2. 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-secret3 key pair via KeyGen. Messages can be encrypted via Encrypt by anyone who also decides what cipher text 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 cipher text provided that the ciphertext's class is contained in the aggregate key via Decrypt.

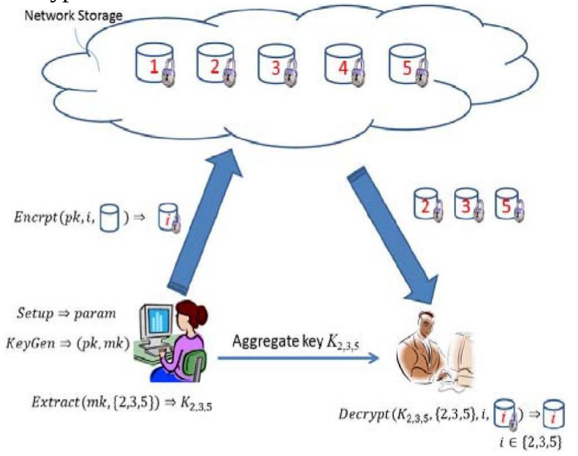
Network Storage
1
2
3
4
5
$Encrpt(pk, i, \cdot) \Rightarrow$
2
3
5
$Setup \Rightarrow param$
$KeyGen \Rightarrow (pk, mk)$
Aggregate key $K_{2,3,5}$
$Extract(mk, \{2,3,5\}) \Rightarrow K_{2,3,5}$
$Decrypt(K_{2,3,5}, \{2,3,5\}, i, \cdot) \Rightarrow$
$i \in \{2,3,5\}$

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

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 cipher text 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 Figure. 2. Suppose Alice wants to share her data m1;m2; … .;m_ on the server. She first performs Setupð1_; nÞ to get param and execute KeyGen to get the public/master-secret key pair. The system parameter 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 mi by Ci ¼ Encrypt. 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 Extractðmsk; SÞ. 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 2 S, Bob downloads Ci (and some needed values in param) from the server. With the aggregate key KS, Bob can decrypt each Ci by DecryptS; i; Ci for each i 2 S.

3. RELATED WORK

This section we compare our basic KAC scheme with other possible solutions on sharing in secure cloud storage.

3.1 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., [11], [12], [13], [14]) 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 [15] proposed a method to generate a tree hierarchy of symmetric-keys 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 [16], [17], [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 like Figure. 3. Each node in the tree represents a secret key, while the leaf nodes represents the keys for individual cipher text 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 nonleaf node can derive the keys of its descendant nodes.
In Figure. 3a, if Alice wants to share all the files in the "personal" category, she only needs to grant the key for the node "personal," which automatically grants the delegate the keys of all the descendant nodes ("photo," "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. 3b, 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 delegate in our example, the

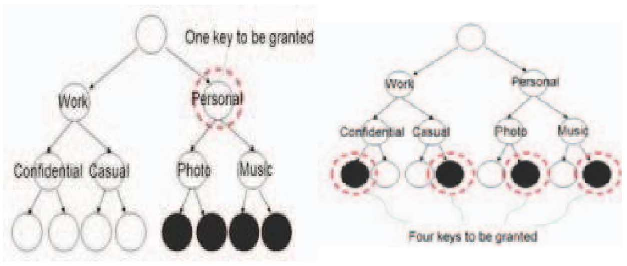

Figure.3. Compact key is not always possible for a fixed hierarchy.

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.

3.2 Compact Key in Symmetric-Key Encryption

Motivated by the same problem of supporting flexible hierarchy in decryption power delegation (but in symmetric- key setting), Benaloh et al. [8] presented an encryption scheme which is originally proposed for concisely transmitting large number of keys in broadcast scenario [18]. The construction is simple and we briefly review its key derivation process here for a concrete description of what are the desirable properties we want to achieve. The derivation of the key for a set of classes (which is a subset of all possible ciphertext classes) is as follows: A composite modulus N ¼ p _ q is chosen where p and q are two large random primes. A master-secret key Y is chosen at random from ZZ_N. Each class is associated with a distinct prime ei. All these prime numbers can be put in the public system parameter.5 A constant-size key for set S0 can be generated.

3.3 Compact Key in Identity-Based Encryption (IBE)

IBE (e.g., [20], [21], [22]) 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 in IBE which holds a master-secret key and issues a secret key to each user with respect to the user identity. The encryption can take the public parameter and a user identity to encrypt a message.
The recipient can decrypt this cipher text by his secret key. Gout et al. [23], [9] tried to build IBE with key aggregation. One of their schemes [23] assumes random oracles but another [9] 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 [23], [9] comes at the expense of OðnÞ sizes for both cipher texts 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 cipher text size, and their security holds in the standard model. In fuzzy IBE [21], one single compact secret key can decrypt cipher texts 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.

3.4 Other Encryption Schemes

Attribute-based encryption (ABE) [10], [24] allows each cipher text to be associated with an attribute, and the master-secret key holder can extract a secret key for a policy of these attributes so that a ciphertext can be decrypted by this key if its associated attribute conforms to the policy. For example, with the secret key for the policy 2 _ 3 _ 6, one can decrypt cipher text tagged with class 2, 3, 6, or 8. However, the major concern in ABE is collusion resistance but not the compactness of secret keys. Indeed, the size of the key often increases linearly with the number of attributes it encompasses, or the cipher text-size is not constant (e.g., [25]). To delegate the decryption power of some cipher texts without sending the secret key to the delegate, a useful primitive is proxy re-encryption (PRE) (e.g., [26], [27], [28], [29]). A PRE scheme allows Alice to delegate to the server (proxy) the ability to convert the cipher texts encrypted under her public-key into ones for Bob. PRE is well known
to have numerous applications including cryptographic file system [30]. Nevertheless, Alice has to trust the proxy that it only converts cipher texts according to her instruction, which is what we want to avoid at the first place. Even worse, if the proxy colludes with Bob, some form of Alice's secret key can be recovered which can decrypt Alice's (convertible) ciphertexts without Bob's further help. That also means that the transformation key of proxy should be well protected. Using PRE just moves the secure key storage requirement from the delegatee to the proxy. It is, thus, undesirable to let the proxy reside in the storage server. That will also be inconvenient since every decryption requires separate interaction with the proxy.

4. 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 3.1. This is used in the complete sub tree 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 Figure. 3), and thus can support up to 2h cipher text classes, a selected part of which is intended for an authorized delegate.
In an ideal case as depicted in Figure. 3a, the delegate can be granted the access to 2hs classes with the possession of only one key, where hs is the height of a certain sub tree (e.g., hs ¼ 2 in Figure. 3a). On the other hand, to decrypt cipher texts of a set of classes, sometimes the delegate may have to hold a large number of keys, as depicted in Figure. 3b 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 cipher text classes, and the delegate of concern is entitled to a portion r of them. That is, r is the delegation ratio, the ratio of the delegated cipher text classes to the total classes. 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" for different delegates. For each combination of r and h, we randomly generate 104 different combinations of classes to be delegated, and the output key set size na is the average over random delegations. We tabulate the results in Table 2, where h ¼ 16; 18; 20 respectively.6 For a given h, na increases with the delegation ratio r until r reaches 70%. An amazing fact is that, the ratio of na to Nð¼ 2hþ1 _ 1Þ, the total number of keys in the hierarchy (e.g., N ¼ 15 in Figure. 3), appears to be only determined by r but irrelevant of h. This is because when the number of cipher text classes (2h) is large and the delegation ratio (r) is fixed, this kind of random delegation achieves roughly the same key assignment ratios (na=N). Thus, for the same r, na grows exponentially with h. We can easily estimate how many keys we need to assign when we are given r and h. We then turn our focus to the compression7 factor F for a certain h, i.e., the average number of delegated classes that each granted key can decrypt. Specifically, it is the ratio of the total number of delegated classes (r2h) to the number of granted keys required (na). Certainly, higher compression factor is preferable because it means each granted key can decrypt more cipher texts. Figure. 4a illustrates the relationship between the compression factor and the delegation ratio Somewhat surprisingly, we found that F ¼ 3:2 even for delegation ratio of r ¼ 0:9, and F < 6 for r ¼ 0:95, which deviates from the intuition that only a small number of "powerful" keys are needed for delegating most of the classes. We can only get a high (but still small) compression factor when the delegation ratio is close to 1. A comparison of the number of granted keys between three methods is depicted in Figure. 4b. We can see that if w grant the key one by one, the number of granted keys would be equal to the number of the delegated cipher text 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. 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.

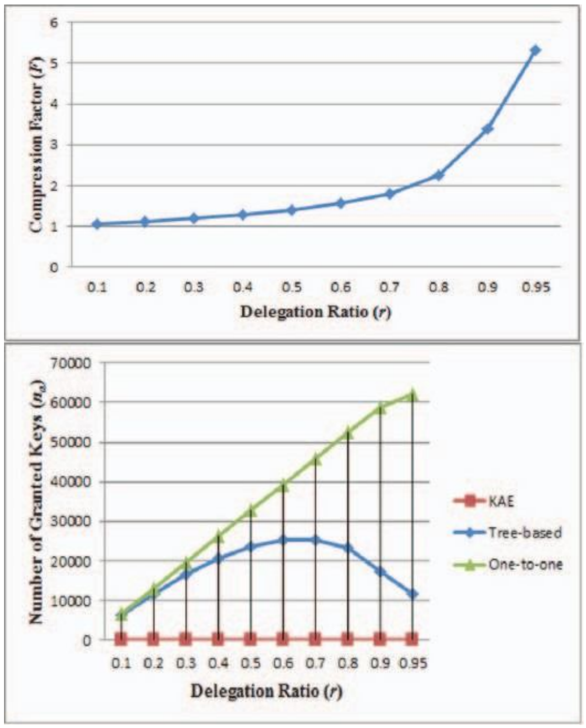

Figure.4.(a) Compression achieved by the tree-based approach for delegating different ratio of the classes. (b) Number of granted keys (n_a) required for different approaches in the case of 65,536 classes of data.

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 sized system parameter. Encryption can be done in constant time, while decryption can be done in group multiplications (or point addition on elliptic curves) with two pairing operations, where S is the set of cipher text classes decrypt able by the granted aggregate key and jSj _ n. As expected, key extraction requires 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. We implemented the basic KAC system 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 cipher text only contains two GG and one GGT elements, we used (symmetric) pairings over Type-A (super singular) 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 Salinas prime, which offers 1,024-bit of discrete-logarithm security. With this Type-A curves setting in PBC, elements of groups GG and GGT take 512 and 1,024 bits to represent, respectively. The test machine is a Sun Ultra Spark system with dual CPU (1,002 MHz) running Solaris, each with 2-GB RAM. The timings reported below are averaged over 100 randomized runs. In our experiment, we take the number of cipher text classes n ¼ 216 ¼ 65,536. The Setup algorithm, while outputting 2n þ 1Þ elements by doing 2n _ 2Þ exponentiations, can be made efficient by preprocessing function offered by PBC, which saves time for exponentiation the same element (g) in the long run. This is the only "low-level" optimization trick we have used. All other operations implemented in a straightforward manner. In particular, we did not exploit the fact that will be exponentiated many times across different encryptions. However, w precomputed its value in the setup stage, such that the encryption can be done without computing any pairing. Our experiment results. 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 I roughly a double of Extract. Note that our experiments dealt with up to 65,536 number of classes (which is also th 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 no confidential storage is limited, one should deploy our schemes using the Type-D pairing bundled with the PBC, which only requires 170-bit to represent an element in GG. For n ¼ 216, 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 cell phone can store more than a dozen of them. But we saved expensive secure storage without the hassle of managing a hierarchy of delegation classes.

5. NEW PATIENT-CONTROLLED ENCRYPTION (PCE)

Motivated by the nationwide effort to computerize America's medical records, the concept of patient-controlled encryption has been studied [8]. In PCE, the health record is decomposed into a hierarchical representation based on the use of different ontology's, and patients are the parties who generate and store secret keys. When there is a need for a healthcare personnel to access part of the record, a patient will release the secret key for the concerned part of the record. In the work of Benelux et al. [8], three solutions have been provided, which are symmetric-key PCE for fixed hierarchy (the "folklore" tree-based method in Section 3.1), public-key PCE for fixed hierarchy (the IBE analog of the folklore method, as mentioned in Section 3.1), and RSA based symmetric-key PCE for "flexible hierarchy" (which is the "set membership" access policy as we explained). Our work provides a candidate solution for the missing piece, public-key PCE for flexible hierarchy, which the existence of an efficient construction was an open question. Any patient can either define her own hierarchy according to her need, or follow the set of categories suggested by the electronic medical record system she is using, such as "clinic visits," "x-rays," "allergies," "medications," and so on. When the patient wishes to give access rights to her doctor, she can choose any subset of these categories and issue a single key, from which keys for all these categories can be computed. Thus, we can essentially use any hierarchy we choose, which is especially useful when the hierarchy can be complex. Finally, one healthcare personnel deals with many patients and the patient record is possible stored in cloud storage due to its huge size (e.g., high-resolution medical imaging employing x-ray), compact key size and easy key management are of paramount importance.

6. 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 paper, we consider how to "compress" secret keys in public-key cryptosystems which support delegation of secret keys for different cipher text 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 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 as we described in Section 4.2. Although the parameter can be downloaded with cipher texts, it would be better if its size is independent of the maximum number of cipher text 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 [22], [34] 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.
11. S.G. Akl and P.D. Taylor, "Cryptographic Solution to a Problem of Access Control in a Hierarchy," ACM Trans. Computer Systems, vol. 1, no. 3, pp. 239-248, 1983.
12. G.C. Chick and S.E. Tavares, "Flexible Access Control with Master Keys," Proc. Advances in Cryptology (CRYPTO '89), vol. 435, pp. 316-322, 1989.
13. W.-G. Tzeng, "A Time-Bound Cryptographic Key Assignment Scheme for Access Control in a Hierarchy," IEEE Trans. Knowledge and Data Eng., vol. 14, no. 1, pp. 182-188, Jan./Feb. 2002.
14. 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.
15. R.S. Sandhu, "Cryptographic Implementation of a Tree Hierarchy for Access Control," Information Processing Letters, vol. 27, no. 2, pp. 95-98, 1988.Y. Sun and K.J.R. Liu, "Scalable Hierarchical Access Control in Secure Group Communications," Proc. IEEE INFOCOM '04, 2004.
16. Q. Zhang and Y. Wang, "A Centralized Key Management Scheme for Hierarchical Access Control," Proc. IEEE Global Telecomm. Conf. (GLOBECOM '04), pp. 2067-2071, 2004.
17. J. Benaloh, "Key Compression and Its Application to Digital Fingerprinting," technical report, Microsoft Research, 2009.
18. 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.
19. D. Boneh and M.K. Franklin, "Identity-Based Encryption from th Weil Pairing," Proc. Advances in Cryptology (CRYPTO '01), vol. 2139, pp. 213-229, 2001.
20. A. Sahai and B. Waters, "Fuzzy Identity-Based Encryption," Proc. 22nd Int'l Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT '05), vol. 3494, pp. 457-473, 2005.
21. 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.
22. F. Guo, Y. Mu, and Z. Chen, "Identity-Based Encryption: How to Decrypt Multiple Ciphertexts Using a Single Decryption Key," Proc. Pairing-Based Cryptography Conf. (Pairing '07), vol. 4575, pp. 392-406, 2007.
23. 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,
24. T. Okamoto and K. Takashima, "Achieving Short Cipher texts 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.
25. R. Canetti and S. Hohenberger, "Chosen-Ciphertext Secure Proxy Re-Encryption," Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), pp. 185-194, 2007.
26. C.-K. Chu and W.-G. Tzeng, "Identity-Based Proxy Re-encryption without Random Oracles," Proc. Information Security Conf. (ISC '07), vol. 4779, pp. 189-202, 2007.
27. C.-K. Chu, J. Weng, S.S.M. Chow, J. Zhou, and R.H. Deng, "Conditional Proxy Broadcast Re-Encryption," Proc. 14th Australasian Conf. Information Security and Privacy (ACISP '09), vol. 5594, pp. 327-342, 2009.
28. 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.
29. 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.
30. D. Boneh, C. Gentry, and B. Waters, "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys," Proc. Advances in Cryptology Conf. (CRYPTO '05), vol. 3621, pp. 258-275, 2005.
31. L.B. Oliveira, D. Aranha, E. Morais, F. Daguano, J. Lopez, and R Dahab, "TinyTate: Computing the Tate Pairing in Resource- Constrained Sensor Nodes," Proc. IEEE Sixth Int'l Symp. Network Computing and Applications (NCA '07), pp. 318-323, 2007.
32. D. Naor, M. Naor, and J. Lotspiech, "Revocation and Tracing Schemes for Stateless Receivers," Proc. Advances in Cryptology Conf. (CRYPTO '01), pp. 41-62, 2001.
33. G.C. Chick and S.E. Tavares, "Flexible Access Control with Master Keys," Proc. Advances in Cryptology (CRYPTO '89), vol. 435, pp. 316-322, 1989.