

# Information Security for Scalable Data Sharing

Mangali Gopal, K.Vinay Kumar, A.Atchyuta Rao  
Gokul institute of technology and sciences

**Abstract** - Information sharing is an important functionality in cloud storage. Here, In this paper, we show how to share data securely, efficiently, and flexibly. A new public-key cryptosystems is demonstrated which produce cipher texts with constant-size such that efficient delegation of decryption rights for any set of ciphertexts are possible. The novelty is that one can aggregate any set of secret keys and make them as a single key, but encompassing the power of all the keys being aggregated. That is, the secret key holder can generate a constant-size aggregate key for flexible choices of ciphertext set in cloud storage, but the other encrypted files with cipher text outside the set can be secure. This compact aggregate key can be sent to others or be stored in a smart card with very limited secure storage. Privacy analysis of our schemes in the standard model and describe other application of our proposed schemes is provided. In particular, our system gives the first public-key patient-controlled encryption for flexible hierarchy, which was yet to be known.

**Index Terms** - Information Security, Privacy preserving, Data Encryption, patient-controlled encryption

## I. INTRODUCTION

We see the rise in demand for data outsourcing, which assists in the strategic management of corporate data in Cloud storage, and 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 25GB. Along with the current wireless technology, users can access almost all of their data by a mobile phone from any place.

In Cloud Storage, data from different clients can be hosted on separate virtual machines (VMs) but reside on a single physical machine. Data in a destined VM could be stolen by instantiating another VM co-resident with the target one. To know about the data availability, 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. 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, with proven security re- lied 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.

Let us consider an example of data sharing through dropbox. We consider that Alice keeps 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 here is how to delegate the decryption rights for these photos to Bob. An option that Alice has is to securely send Bob the secret keys involved. Generally, there are two 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.

Clearly, the first method is inadequate since all un- chosen data may be also leaked to Bob. For other one, there are practical concerns on efficiency. We should have separate key for every photo, 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 for these generally increase with the number of the decryption keys to be shared. In other words, it is very expensive.

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.

The best available solution for this 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. Example, we can not 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.

## II. INFORMATION ENCRYPTION

### Framework

Mainly, a key-aggregate encryption scheme has 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. Master secret can be used by the data owner to generate an aggregate decryption key for a set of ciphertext classes. 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 mechanism.

- Setup( $1\lambda, n$ ): executed by the data owner to setup an account on an untrusted server. On input a security level parameter  $1\lambda$  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  $i \in S$ .

### Encrypted Information Sharing

An application of KAC is data sharing. 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 Figure 2. Suppose Alice wants to share her data  $m_1, m_2, \dots, m_v$  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  $C_i = \text{Encrypt}(pk, i, m_i)$ . Finally, the encrypted data are sent to 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 getting the aggregate key, Bob can download the data that he has access with. That is, for each  $i \in S$ , Bob downloads  $C_i$  (and some needed values in  $param$ ) from the server. With the aggregate key  $KS$ , Bob can decrypt each  $C_i$  by Decrypt( $KS, S, i, C_i$ ) for each  $i \in S$ .

## III. RELATED WORK

Here we compare our basic KAC scheme with other possible solutions on sharing in secure cloud storage. We summarize our comparisons in Table 1.

### Cryptographic Keys

Here we discuss the most relevant study in the cryptography/security. Cryptographic key assignment schemes aim to reduce 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. Just granting the parent key implicitly grants all the keys of its descendant nodes.. 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. 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.

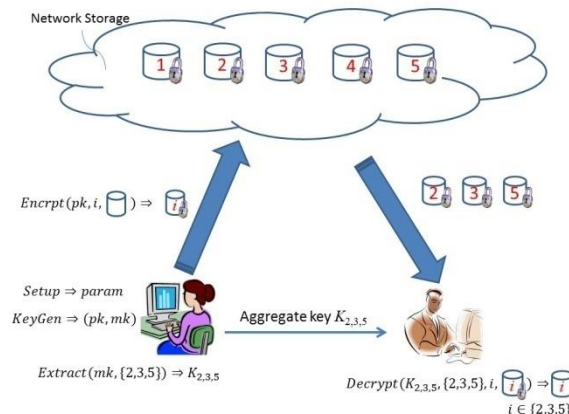


Figure 2

Consider the below figure3, 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.

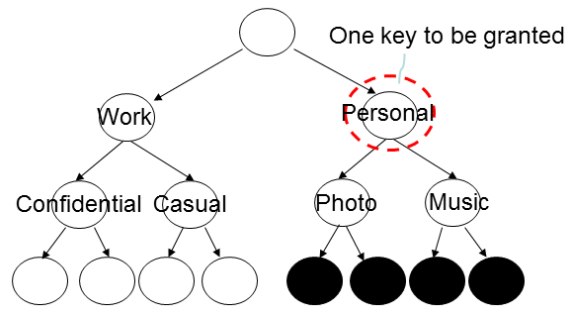


Fig. 3a

Table I : Comparisons between our basic KAC scheme and other related schemes

	Decryption key size	Ciphertext size	Encryption type
Key assignment schemes for a predefined hierarchy (e.g., [7])	most likely non-constant (depends on the hierarchy)	constant	symmetric or public-key
Symmetric-key encryption with Compact Key (e.g., [8])	constant	constant	symmetric-key
IBE with Compact Key (e.g., [9])	constant	non-constant	public-key
Attribute-Based Encryption (e.g., [10])	non-constant	constant	public-key
KAC	constant	constant	public-key

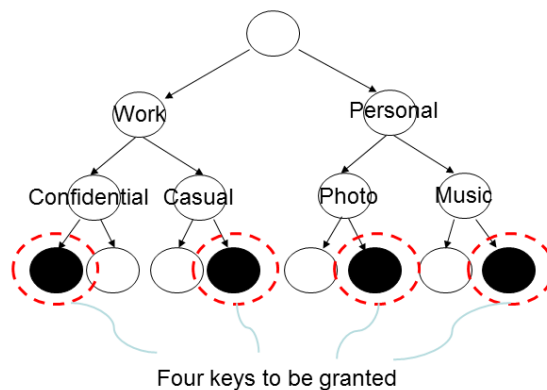


Fig 3b

Fig. 3. Compact key is not always possible for a fixed Hierarchy

In Figure 3(a), 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 delegatee 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. But in general cases, it is difficult. As shown in Figure 3(b), 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, the number of granted secret keys becomes the same as the number of classes.

**Compact Key in Symmetric-Key Encryption**

Benaloh et al. 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  $N = p \cdot q$  is chosen where  $p$  and  $q$  are two large random primes. A master secret key  $Y$  is chosen at random from  $Z^*$ . Each class is associated with a distinct prime  $e_i$ . All these prime numbers can be put in the public system parameter. A constant-size key for set  $S_0$  can be generated. As a concrete example, a key for classes represented by  $e_1, e_2, e_3$  can be generated as  $Y(1/e_1.e_2.e_3)$  from which,  $Y1/e_1, Y1/e_2, Y1/e_3$  can be easily derived.

This approach achieves similar properties and performances as our schemes. However, it is designed for the symmetric-key setting instead. The encryptor needs to get the corresponding secret keys to encrypt data, which is not suitable for many applications. Since their method is used to generate a secret value rather than a pair of public/secret keys, it is unclear how to apply this idea for public-key encryption scheme.

#### IV. CONCRETE CONSTRUCTIONS OF CRYPTOSYSTEM

Let  $G$  and  $G_T$  be two Cyclic groups of prime order  $p$  and  $e:G \times G \rightarrow G_T$  be a map with the following properties.

- Bilinear:  $\forall g_1, g_2 \in G, a, b \in \mathbb{Z}, \hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$ .
- Bilinear:  $\forall g_1, g_2 \in G, a, b \in \mathbb{Z}, \hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$ .
- Non-degenerate: for some  $g \in G, \hat{e}(g, g) \neq 1$ .

$G$  is a bilinear group if all the options involved above are efficiently computable.. many classes of elliptic curves feature bilinear groups.

##### Basic Construction

Although some scheme supports constant-size secret keys, every key only has the power for decrypting ciphertexts associated to a particular index. We thus need to devise a new Extract algorithm and the corresponding Decrypt algorithm.

- Setup( $1^\lambda, n$ ): Randomly pick a bilinear group  $G$  of prime order  $p$  where  $2^\lambda \leq p \leq 2^{\lambda+1}$ , a generator  $g \in G$  and  $\alpha \in_R \mathbb{Z}_p$ . Compute  $g_i = g^{\alpha^i} \in G$  for  $i = 1, \dots, n, n+2, \dots, 2n$ . Output the system parameter as  $\text{param} = (g, g_1, \dots, g_n; g_{n+2}, \dots, g_{2n})$  ( $\alpha$  can be safely deleted after Setup).

- KeyGen(): Pick  $\gamma \in_R \mathbb{Z}_p$ , output the public and aster-secret key pair:  $(\text{pk} = v = g^\gamma, \text{msk} = \gamma)$ .

- Encrypt( $\text{pk}; i; m$ ): For a message  $m \in G_T$  and an index  $i \in \{1, 2, \dots, n\}$ , randomly pick  $t \in_R \mathbb{Z}_p$  and compute the cipher text  $\mathcal{C} = \langle g^t, (vg_i)^t, m \cdot \hat{e}(g_1, g_n)^t \rangle$ .

- Extract( $\text{msk} = \gamma, \mathcal{S}$ ): for the set  $\mathcal{S}$  of indices  $j$ 's, the aggregate key is computed as  $K_S = \prod_{j \in \mathcal{S}} g_{n+1-j}^\gamma$  since  $\mathcal{S}$  does not include  $\hat{e}(g_1, g_n)^t$ . can always be retrieved from param.

- Decrypt( $K_S, \mathcal{S}, i, \mathcal{C} = \langle c_1, c_2, c_3 \rangle$ ): If  $i \notin \mathcal{S}$ , output  $\perp$ . Otherwise, return the message:  $m = c_3 \cdot \hat{e}(K_S \cdot \prod_{j \in \mathcal{S}, j \neq i} g_{n+1-j+i}, c_1) / \hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j}, c_2)$ .

For the data owner, with the knowledge of  $\gamma$ , the term  $\hat{e}(g_1, g_n)^t$  can be easily recovered by  $\hat{e}(c_1, g_n)^\gamma = \hat{e}(g^t, g_n)^\gamma = \hat{e}(g_1, g_n)^t$ .

For correctness, we can see that

$$\begin{aligned} & c_3 \cdot \hat{e}(K_S \cdot \prod_{j \in \mathcal{S}, j \neq i} g_{n+1-j+i}, c_1) / \hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j}, c_2) \\ = & c_3 \cdot \frac{\hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j}^\gamma \cdot \prod_{j \in \mathcal{S}, j \neq i} g_{n+1-j+i}, g^t)}{\hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j}, (vg_i)^t)} \\ = & c_3 \cdot \frac{\hat{e}(\prod_{j \in \mathcal{S}, j \neq i} g_{n+1-j+i}, g^t) / \hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j}, g_i^t)}{\hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j+i}, g^t) / \hat{e}(g_{n+1}, g^t)} \\ = & c_3 \cdot \frac{\hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j+i}, g^t)}{\hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j+i}, g^t)} \\ = & m \cdot \hat{e}(g_1, g_n)^t / \hat{e}(g_{n+1}, g^t) = m. \end{aligned}$$

##### Extension Of Public-Key

If a user needs to classify his ciphertexts into more than  $n$  classes, he can register for additional key pairs  $(\text{pk}_2; \text{msk}_2), \dots, (\text{pk}_\ell; \text{msk}_\ell)$ . Each class now is indexed by a 2-level index in  $\{(i, j) | 1 \leq i \leq \ell, 1 \leq j \leq n\}$  and the number of classes is increased by  $n$  for each added key.

Figure 4. Figure 4 shows the flexibility of our approach. We achieve “local aggregation”, which means the secret keys under the same branch can always be aggregated. We use a quaternary tree for the last level just for better illustration of our distinctive feature. Our advantage is still preserved when compared with quaternary trees in hierarchical approach, in which the latter either delegates the decryption power for all 4 classes (if the key for their parent class is delegated) or the number of keys will be the same as the number of classes. For our approach, at most 2 aggregate keys are needed in our example.

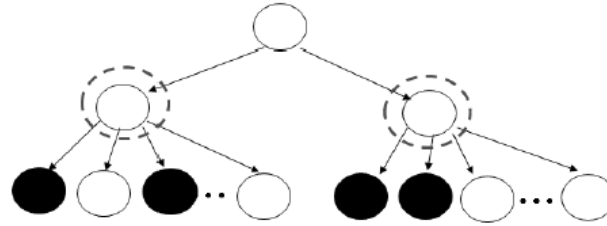


Fig. 4. Key assignment in our approach

- Setup and KeyGen: Same as the basic construction
- Extend(pk<sub>l</sub>; msk<sub>l</sub>): Execute KeyGen() to get  $(v_{l+1}, \gamma_{l+1}) \in \mathbb{G} \times \mathbb{Z}_p$ , output the public and extended master-secret keys as  $\text{pk}_{l+1} = (\text{pk}_l, v_{l+1}), \text{msk}_{l+1} = (\text{msk}_l, \gamma_{l+1})$ .
- Encrypt(pk<sub>l</sub>, (a, b), m): Let  $\text{pk}_l = \{v_1, \dots, v_l\}$ . For an index  $(a, b), 1 \leq a \leq l, 1 \leq b \leq n$ , pick  $t \in_R \mathbb{Z}_p$ , output the ciphertext as  $\mathcal{C} = \langle g^t, (v_a g_b)^t, m \cdot \hat{e}(g_1, g_n)^t \rangle$ .
- Extract(msk<sub>l</sub>, S<sub>l</sub>): Let  $\text{msk}_l = \{\gamma_1, \gamma_2, \dots, \gamma_l\}$ . For a set S<sub>l</sub> of indices  $(i, j), 1 \leq i \leq l, 1 \leq j \leq n$ , get  $g_{n+1-j} = g^{\alpha^{n+1-j}}$  from param, output:

$$K_{S_l} = \left( \prod_{(1,j) \in S_l} g_{n+1-j}^{\gamma_1}, \prod_{(2,j) \in S_l} g_{n+1-j}^{\gamma_2}, \dots, \prod_{(l,j) \in S_l} g_{n+1-j}^{\gamma_l} \right).$$

- Decrypt(K<sub>S<sub>l</sub></sub>, S<sub>l</sub>, (a, b), C): If  $(a, b) \notin S_l$ , output  $\perp$ . Otherwise, let  $K_{S_l} = (d_1, \dots, d_l)$  and  $\mathcal{C} = \langle c_1, c_2, c_3 \rangle$ . Output the message:

$$m = \frac{c_3 \cdot \hat{e}(d_a \cdot \prod_{(a,j) \in S_l, j \neq b} g_{n+1-j+b}, c_1)}{\hat{e}(\prod_{(a,j) \in S_l} g_{n+1-j}, c_2)}.$$

Just like the basic construction, the decryption can be done more efficiently with the knowledge of i's. Correctness is not much more difficult to see:

$$\begin{aligned} & c_3 \cdot \hat{e}(d_a \cdot \prod_{(a,j) \in S_l, j \neq b} g_{n+1-j+b}, c_1) \\ & / \hat{e}(\prod_{(a,j) \in S_l} g_{n+1-j}, c_2) \\ = & c_3 \cdot \hat{e}(\prod_{(a,j) \in S_l} g_{n+1-j}^{\gamma_a} \cdot \prod_{(a,j) \in S_l, j \neq b} g_{n+1-j+b}, g^t) \\ & / \hat{e}(\prod_{(a,j) \in S_l} g_{n+1-j}, (v_a g_b)^t) \\ = & c_3 \cdot \hat{e}(\prod_{(a,j) \in S_l, j \neq b} g_{n+1-j+b}, g^t) / \hat{e}(\prod_{(a,j) \in S_l} g_{n+1-j}, g_b^t) \\ = & m \cdot \hat{e}(g_1, g_n)^t / \hat{e}(g_{n+1}, g^t) = m. \end{aligned}$$

## V. ANALYSING PERFORMANCE

### Compression Factors

We consider, 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. Obviously, if **r** = 0, **n<sub>a</sub>** should also be 0, which means no access to any of the classes; if **r** = 100%, **n<sub>a</sub>** 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 **n<sub>a</sub>** may first increase with **r**, and may decrease later. We set **r** = 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  $n_a$  is the average over random delegations.

We tabulate the results in Table 2, where  $h = 16; 18; 20$  respectively. For a given  $h$ ,  $n_a$  increases with the delegation ratio  $r$  until  $r$  reaches  $\sim 70\%$ . An amazing fact is that, the ratio of  $n_a$  to  $N (= 2^{h+1} - 1)$ , the total number of keys in the hierarchy (e.g.,  $N = 15$  in Figure 3), appears to be only determined by  $r$  but irrelevant of  $h$ . This is because when the number of ciphertext classes ( $2h$ ) is large and the delegation ratio ( $r$ ) is fixed, this kind of random delegation achieves roughly the same key assignment ratios ( $n_a=N$ ). Thus, for the same  $r$ ,  $n_a$  grows exponentially with  $h$ . We can easily estimate how many keys we need to assign when we are given  $r$  and  $h$ .

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 ( $n_a$ ). Certainly, higher compression factor is preferable because it means each granted key can decrypt more ciphertexts. Figure 5(a) illustrates the relationship between the compression factor and the delegation ratio. Somewhat surprisingly, we found that  $F = 3:2$  even for delegation ratio of  $r = 0:9$ , and  $F < 6$  for  $r = 0:95$ , 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 and a comparison of the number of granted keys between three methods is depicted in Figure 5(b).

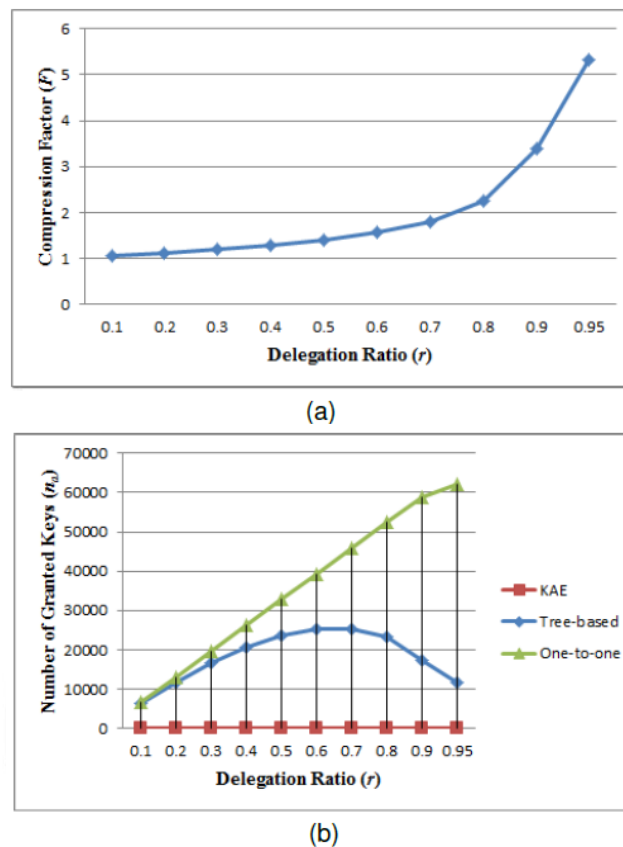


Fig. 5. (a) Compression achieved by the tree-based approach for delegating different ratio of the classes (b) Number of granted keys ( $n_a$ ) required for different approaches in the case of 65536 classes of data

## VI. CONTROLLED ENCRYPTION- NEW PATIENT

Motivated by the nationwide effort to computerize America’s medical records, the concept of patientcontrolled encryption (PCE) has been studied. In PCE, the health record is decomposed into a hierarchical representation based on the use of different ontologies, 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 Benaloh 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,

$h$	$r$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95
16	$n_a$	6224.8	11772.5	16579.3	20545.8	23520.7	25263.8	25400.1	23252.6	17334.6	11670.2
	$\frac{n_a}{N}$	4.75%	8.98%	12.65%	15.68%	17.94%	19.27%	19.38%	17.74%	13.23%	8.90%
18	$n_a$	24895.8	47076.1	66312.4	82187.1	94078.8	101052.4	101594.8	93025.4	69337.4	46678.8
	$\frac{n_a}{N}$	4.75%	8.98%	12.65%	15.68%	17.94%	19.27%	19.38%	17.74%	13.23%	8.90%
20	$n_a$	99590.5	188322.0	265254.1	328749.5	376317.4	404205.0	406385.1	372085.2	277343.1	186725.4
	$\frac{n_a}{N}$	4.75%	8.98%	12.65%	15.68%	17.94%	19.27%	19.38%	17.74%	13.22%	8.90%

**TABLE 2**  
**Compression ratios for different delegation ratios and tree heights**

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.

## VII. 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 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.

## VIII. 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," in *Applied Cryptography and Network Security – ACNS 2012*, ser. LNCS, vol. 7341. Springer, 2012, pp. 526–543.
- [2] L. Hardesty, "Secure computers aren't so secure," MIT press, 2009, <http://www.physorg.com/news176107396.html>.
- [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, 2013.
- [4] B. Wang, S. S. M. Chow, M. Li, and H. Li, "Storing Shared Data on the Cloud via Security-Mediator," in *International Conference on Distributed Computing Systems - ICDCS 2013*. IEEE, 2013.
- [5] S. S. M. Chow, C.-K. Chu, X. Huang, J. Zhou, and R. H. Deng, "Dynamic Secure Cloud Storage with Provenance," in *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, ser. LNCS, vol. 6805. Springer, 2012, pp. 442–464.
- [6] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," in *Proceedings of Advances in Cryptology - EUROCRYPT '03*, ser. LNCS, vol. 2656. Springer, 2003, pp. 416–432.
- [7] 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.
- [8] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, "Patient Controlled Encryption: Ensuring Privacy of Electronic Medical Records," in *Proceedings of ACM Workshop on Cloud Computing Security (CCSW '09)*. ACM, 2009, pp. 103–114.
- [9] F. Guo, Y. Mu, Z. Chen, and L. Xu, "Multi-Identity Single-Key Decryption without Random Oracles," in *Proceedings of Information Security and Cryptology (Inscrypt '07)*, ser. LNCS, vol. 4990. Springer, 2007, pp. 384–398.
- [10] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*. ACM, 2006, pp. 89–98.
- [11] 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.
- [12] G. C. Chick and S. E. Tavares, "Flexible Access Control with Master Keys," in *Proceedings of Advances in Cryptology – CRYPTO '89*, ser. LNCS, vol. 435. Springer, 1989, pp. 316–322.