2-2014

Key-Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage

Cheng-Kang CHU
Institute for Infocomm Research, Singapore

Sherman S. M. CHOW
University of Hong Kong

Wen-Guey TZENG
National Chiao Tung University

Jiangying ZHOU
Institute for Infocomm Research, Singapore

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Citation

CHU, Cheng-Kang; CHOW, Sherman S. M.; TZENG, Wen-Guey; ZHOU, Jiangying; and DENG, Robert H.. Key-Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage. (2014). *IEEE Transactions on Parallel and Distributed Systems*. 25, (2), 468-477.
Available at: https://ink.library.smu.edu.sg/sis_research/1938

Key-Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage

Cheng-Kang Chu, Sherman S. M. Chow, Wen-Guey Tzeng, Jianying Zhou, and Robert H. Deng, *Senior Member, IEEE*

Abstract—Data sharing is an important functionality in cloud storage. In this article, 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 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 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.

Index Terms—Cloud storage, data sharing, key-aggregate encryption, patient-controlled encryption

1 Introduction

Cloud 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 25GB (or a few dollars for more than 1TB). 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, e.g., [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 Dropbox[1] 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

- *C.-K. Chu and J. Zhou are with the Cryptography and Security Department at Institute for Infocomm Research, Singapore.*
- *S. S.-M. Chow is with the Department of Information Engineering, Chinese University of Hong Kong.*
- *W.-G. Tzeng is with the Department of Computer Science, National Chiao Tung University, Taiwan.*
- *R. H. Deng is with the School of Information Systems, Singapore Management University.*

- *This work was supported by the Singapore A*STAR project SecDC-112172014.*

[1] http://www.dropbox.com

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 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 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 (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 –

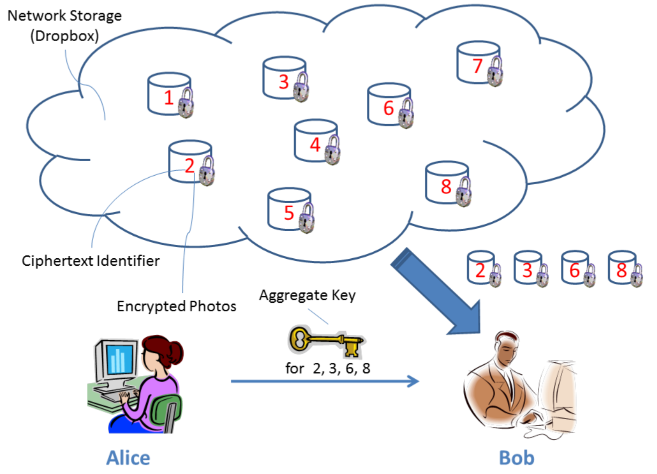


Fig. 1. Alice shares files with identifiers 2, 3, 6 and 8 with Bob by sending him a single aggregate key.

"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.

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 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 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 pre-defined 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 article. All constructions can be proven secure in the standard model. To the best of our knowledge, our aggregation mechanism[2] 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-secret[3] 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[4].

- $\mathsf{Setup}(1^\lambda, n)$: 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 $\mathtt{param}$, 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 $(\mathtt{pk}, \mathtt{msk})$.
- $\mathsf{Encrypt}(\mathtt{pk}, i, m)$: executed by anyone who wants to encrypt data. On input a public-key $\mathtt{pk}$, an index i denoting the ciphertext class, and a message m, it outputs a ciphertext $\mathfrak{C}$.
- $\mathsf{Extract}(\mathtt{msk}, \mathcal{S})$: 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 $\mathtt{msk}$ and a set $\mathcal{S}$ of indices corresponding to different classes, it outputs the aggregate key for set $\mathcal{S}$ denoted by $K_\mathcal{S}$.
- $\mathsf{Decrypt}(K_\mathcal{S}, \mathcal{S}, i, \mathfrak{C})$: executed by a delegatee who received an aggregate key $K_\mathcal{S}$ generated by Extract. On input $K_\mathcal{S}$, the set $\mathcal{S}$, an index i denoting the ciphertext class the ciphertext $\mathfrak{C}$ belongs to, and $\mathfrak{C}$, it outputs the decrypted result m if $i \in \mathcal{S}$.

There are two functional requirements:

- Correctness For any integers λ and n, any set $\mathcal{S} \subseteq \{1, \cdots, n\}$, any index $i \in \mathcal{S}$ and any message m,

$$\begin{aligned}\Pr[\mathsf{Decrypt}(K_\mathcal{S}, \mathcal{S}, i, \mathfrak{C}) = m : \mathtt{param} \leftarrow \mathsf{Setup}(1^\lambda, n),\\ (\mathtt{pk}, \mathtt{msk}) \leftarrow \mathsf{KeyGen}(), \mathfrak{C} \leftarrow \mathsf{Encrypt}(\mathtt{pk}, i, m),\\ K_\mathcal{S} \leftarrow \mathsf{Extract}(\mathtt{msk}, \mathcal{S})] = 1.\end{aligned}$$

- Compactness For any integers λ, n, any set $\mathcal{S}$, any index $i \in \mathcal{S}$ and any message m; $\mathtt{param} \leftarrow \mathsf{Setup}(1^\lambda, n)$, $(\mathtt{pk}, \mathtt{msk}) \leftarrow \mathsf{KeyGen}()$, $K_\mathcal{S} \leftarrow \mathsf{Extract}(\mathtt{msk}, \mathcal{S})$ and $\mathfrak{C} \leftarrow \mathsf{Encrypt}(\mathtt{pk}, i, m)$; $|K_\mathcal{S}|$ and $|\mathfrak{C}|$ only depend on the security parameter λ but independent of the number of classes n.

2.2 Sharing Encrypted Data

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

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 $m_1, m_2, \ldots, m_\nu$ on the server. She first performs $\mathsf{Setup}(1^\lambda, n)$ to get $\mathtt{param}$ and execute KeyGen to get the public/master-secret key pair $(\mathtt{pk}, \mathtt{msk})$. The system parameter $\mathtt{param}$ and public-key $\mathtt{pk}$ can be made public and master-secret key $\mathtt{msk}$ should be kept secret by Alice. Anyone (including Alice herself) can then encrypt each m_i by $\mathfrak{C}_i = \mathsf{Encrypt}(\mathtt{pk}, i, m_i)$. The encrypted data are uploaded to the server.

With $\mathtt{param}$ and $\mathtt{pk}$, people who cooperate with Alice can update Alice's data on the server. Once Alice is willing to share a set $\mathcal{S}$ of her data with a friend Bob, she can compute the aggregate key $K_\mathcal{S}$ for Bob by performing $\mathsf{Extract}(\mathtt{msk}, \mathcal{S})$. Since $K_\mathcal{S}$ 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 $i \in \mathcal{S}$, Bob downloads $\mathfrak{C}_i$ (and some needed values in $\mathtt{param}$) from the server. With the aggregate key $K_\mathcal{S}$, Bob can decrypt each $\mathfrak{C}_i$ by $\mathsf{Decrypt}(K_\mathcal{S}, \mathcal{S}, i, \mathfrak{C}_i)$ for each $i \in \mathcal{S}$.

3 Related Work

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

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

[2] It is obvious that we are *not* proposing an algorithm to compress the decryption key. On one hand, cryptographic keys come from a high-entropy source and are hardly compressible. On the other hand, decryption keys for all possible combinations of ciphertext classes are all in constant-size — information theoretically speaking such compression scheme cannot exist.

[3] We call this as master-secret key to avoid confusion with the delegated key we will explain later.

[4] For simplicity, we omit the inclusion of a decryption algorithm for the original data owner using the master-secret key. In our specific constructions, we will show how the knowledge of the master-secret key allows a faster decryption than using Extract followed by Decrypt.

	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

TABLE 1
Comparisons between our basic KAC scheme and other related schemes

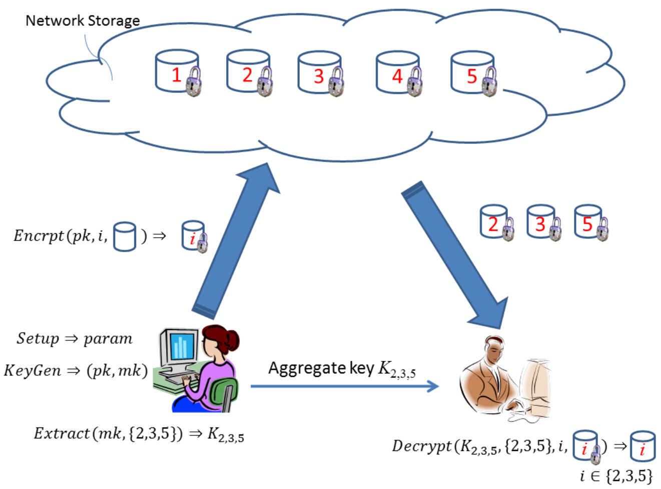

Network Storage
1
2
3
4
5
$Encrpt(pk, i, \ ) \Rightarrow$
2
3
5
$Setup \Rightarrow param$
$KeyGen \Rightarrow (pk, mk)$
Aggregate key $K_{2,3,5}$
$Extract(mk, \{2,3,5\}) \Rightarrow K_{2,3,5}$
$Decrypt(K_{2,3,5}, \{2,3,5\}, i, \ ) \Rightarrow$
$i \in \{2,3,5\}$

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

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 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.

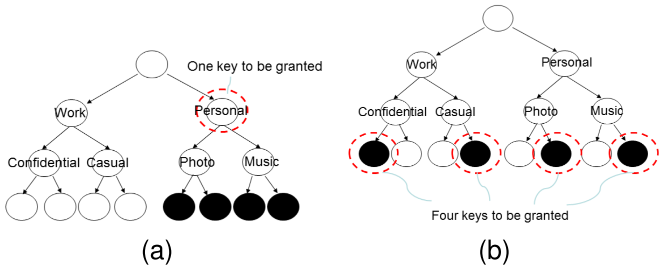


Fig. 3. Compact key is not always possible for a fixed hierarchy

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.

However, it is still difficult for general cases. 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 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*.

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 $N = p \cdot q$ is chosen where p and q are two large random primes. A master-secret key Y is chosen at random from $\mathbb{Z}_N^*$. Each class is associated with a distinct prime e_i. All these prime numbers can be put in the public system parameter[5] A constant-size key for set $\mathcal{S}'$ can be generated (with the knowledge of $\phi(N)$) as $k_{\mathcal{S}'} = Y^{1/\prod_{j \in \mathcal{S}'}(e_j)} \bmod N$. For those who have been delegated the access rights for $\mathcal{S}$ where $\mathcal{S}' \subset \mathcal{S}$, $k_{\mathcal{S}'}$ can be computed by $k_{\mathcal{S}}^{\prod_{j \in \mathcal{S} \setminus \mathcal{S}'}(e_j)}$. As a concrete example, a key for classes represented by e_1, e_2, e_3 can be generated as $Y^{1/(e_1 \cdot e_2 \cdot e_3)}$, from which each of Y^{1/e_1}, Y^{1/e_2}, Y^{1/e_3} can easily be derived (while providing no information about keys for any other class, say, e_4). 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.

Finally, we note that there are schemes which try to reduce the key size for achieving authentication in symmetric-key encryption, e.g., [19]. However, sharing of decryption power is not a concern in these schemes.

3.3 Compact Key in Identity-Based Encryption

Identity-based encryption (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 (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.* [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 $O(n)$ 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 [21], 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.

3.4 Other Encryption Schemes

Attribute-based encryption (ABE) [10], [24] allows each ciphertext 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 $(2 \vee 3 \vee 6 \vee 8)$, one can decrypt ciphertext 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 ciphertext-size is not constant (e.g., [25]).

To delegate the decryption power of some ciphertexts without sending the secret key to the delegatee, 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 ciphertexts 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 ciphertexts 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 Concrete Constructions of KAC

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups of prime order p and $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a map with the following properties:

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

$\mathbb{G}$ is a bilinear group if all the operations involved above are efficiently computable. Many classes of elliptic curves feature bilinear groups.

4.1 A Basic Construction

The design of our basic scheme is inspired from the collusion-resistant broadcast encryption scheme proposed by Boneh *et al.* [31]. Although their scheme supports constant-size secret keys, every key only has

[5] Another way to do this is to apply hash function to the string denoting the class, and keep hashing repeatedly until a prime is obtained as the output of the hash function.

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 $\mathbb{G}$ of prime order p where $2^\lambda \leq p \leq 2^{\lambda+1}$, a generator $g \in \mathbb{G}$ and $\alpha \in_R \mathbb{Z}_p$. Compute $g_i = g^{\alpha^i} \in \mathbb{G}$ for $i = 1, \cdots, n, n+2, \cdots, 2n$. Output the system parameter as $\mathtt{param} = \langle g, g_1, \cdots, g_n, g_{n+2}, \cdots, g_{2n} \rangle$ (α can be safely deleted after Setup).
  Note that each ciphertext class is represented by an index in the integer set $\{1, 2, \cdots, n\}$, where n is the maximum number of ciphertext classes.
- KeyGen(): Pick $\gamma \in_R \mathbb{Z}_p$, output the public and master-secret key pair: $(\mathtt{pk} = v = g^\gamma, \mathtt{msk} = \gamma)$.
- Encrypt($\mathtt{pk}, i, m$): For a message $m \in \mathbb{G}_T$ and an index $i \in \{1, 2, \cdots, n\}$, randomly pick $t \in_R \mathbb{Z}_p$ and compute the ciphertext as $\mathfrak{C} = \langle g^t, (vg_i)^t, m \cdot \hat{e}(g_1, g_n)^t \rangle$.
- Extract($\mathtt{msk} = \gamma, \mathcal{S}$): For the set $\mathcal{S}$ of indices j's, the aggregate key is computed as $K_\mathcal{S} = \prod_{j \in \mathcal{S}} g^\gamma_{n+1-j}$.
  Since $\mathcal{S}$ does not include 0, $g_{n+1-j} = g^{\alpha^{n+1-j}}$ can always be retrieved from $\mathtt{param}$.
- Decrypt($K_\mathcal{S}, \mathcal{S}, i, \mathfrak{C} = \langle c_1, c_2, c_3 \rangle$): If $i \notin \mathcal{S}$, output $\perp$. Otherwise, return the message: $m = c_3 \cdot \hat{e}(K_\mathcal{S} \cdot \prod_{j \in \mathcal{S}, j \neq i} g_{n+1-j+i}, c_1)/\hat{e}(\prod_{j \in \mathcal{S}} g_{n+1-j}, c_2)$.
  For the data owner, with the knowledge of γ, 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

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

4.1.1 Performance

For encryption, the value $\hat{e}(g_1, g_n)$ can be pre-computed and put in the system parameter. On the other hand, we can see that decryption only takes two pairings while only one of them involves the aggregate key. That means we only need one pairing computation within the security chip storing the (secret) aggregate key. It is fast to compute a pairing nowadays, even in resource-constrained devices. Efficient software implementations exist even for sensor nodes [32].

4.1.2 Discussions

The "magic" of getting constant-size aggregate key and constant-size ciphertext simultaneously comes from the linear-size system parameter. Our motivation is to reduce the *secure* storage and this is a trade-off between two kinds of storage. The parameter can be placed in non-confidential local storage or in a cache provided by the service company. They can also be fetched on demand, as not all of them are required in all occasions.

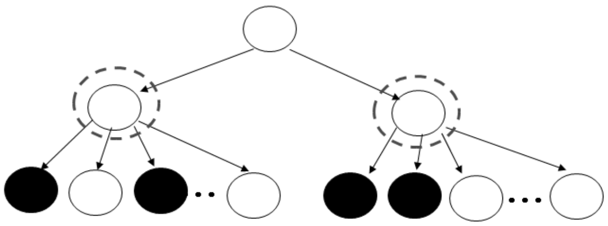


Fig. 4. Key assignment in our approach

The system parameter can also be generated by a trusted party, shared between all users and even hard-coded to the user program (and can be updated via "patches"). In this case, while the users need to trust the parameter-generator for securely erasing any ephemeral values used, the access control is still ensured by a cryptographic mean instead of relying on some server to restrict the accesses honestly.

4.2 Public-Key Extension

If a user needs to classify his ciphertexts into more than n classes, he can register for additional key pairs $(\mathtt{pk}_2, \mathtt{msk}_2), \cdots, (\mathtt{pk}_\ell, \mathtt{msk}_\ell)$. 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.

Since the new public-key can be essentially treated as a new user, one may have the concern that key aggregation across two independent users is not possible. It seems that we face the problem of hierarchical solution as reviewed in Section 1, but indeed, we still achieve shorter key size and gain flexibility as illustrated in 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.

Below we give the details on how encryption and decryption work when the public-key is extended, which is similar to the "$\sqrt{n}$-approach" [31].

- Setup and KeyGen: Same as the basic construction.
- Extend($\mathtt{pk}_l, \mathtt{msk}_l$): Execute KeyGen() to get $(v_{l+1}, \gamma_{l+1}) \in \mathbb{G} \times \mathbb{Z}_p$, output the extended public and master-secret keys as $\mathtt{pk}_{l+1} = (\mathtt{pk}_l, v_{l+1}), \mathtt{msk}_{l+1} = (\mathtt{msk}_l, \gamma_{l+1})$.

- $\mathsf{Encrypt}(\mathtt{pk}_l, (a,b), m)$: Let $\mathtt{pk}_l = \{v_1, \cdots, v_l\}$. 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 $\mathfrak{C} = \langle g^t, (v_a g_b)^t, m \cdot \hat{e}(g_1, g_n)^t \rangle$.
- $\mathsf{Extract}(\mathtt{msk}_l, \mathcal{S}_l)$: Let $\mathtt{msk}_l = \{\gamma_1, \gamma_2, \cdots, \gamma_l\}$. For a set $\mathcal{S}_l$ 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 $\mathtt{param}$, output:

$$\begin{aligned} K_{\mathcal{S}_l} &= (\prod_{(1,j) \in \mathcal{S}_l} g_{n+1-j}^{\gamma_1}, \prod_{(2,j) \in \mathcal{S}_l} g_{n+1-j}^{\gamma_2}, \\ & \qquad \cdots, \prod_{(l,j) \in \mathcal{S}_l} g_{n+1-j}^{\gamma_l}). \end{aligned}$$

- $\mathsf{Decrypt}(K_{\mathcal{S}_l}, \mathcal{S}_l, (a,b), \mathfrak{C})$: If $(a,b) \notin \mathcal{S}_l$, output $\perp$. Otherwise, let $K_{\mathcal{S}_l} = (d_1, \cdots, d_l)$ and $\mathfrak{C} = \langle c_1, c_2, c_3 \rangle$. Output the message:

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

  Just like the basic construction, the decryption can be done more efficiently with the knowledge of γ_i's.

Correctness is not much more difficult to see:

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

We can also prove the semantic security of this extended scheme. The proof is very similar to that for the basic scheme and therefore is omitted. The public-key of our CCA construction to be presented below can also be extended using the same Extend algorithm.

4.2.1 Discussions

To make the best out of our extended scheme (i.e., to make the key size as small as possible), we suggest that the ciphertext classes for different purposes should be corresponded to different public-keys. This is reasonable in practice and does not contradict our criticism on hierarchical methods that an efficient assignment of hierarchy requires a priori knowledge on what to be shared. Using our example, $\mathtt{pk}_1$ and $\mathtt{pk}_2$ correspond to "personal" and "work". It is likely to have many subcategories under either of them but it may not be equally likely to share both of them (if the user does not gossip about office drama with friends and do not expose party photos to colleagues). Another example, say a user's categorization include "music" and "game". One day she becomes a graduate student and needs to publish, and therefore find the new need to add a category "paper", which is probably independent of "music" and "game".

4.2.2 Other Implication

This key extension approach can also be seen as a key update process. In case a secret value is compromised, we can replace the compromised $\mathtt{pk}_1$ with a new key $\mathtt{pk}_2$. The small aggregate key size minimizes the communication overhead for transferring the new key.

5 Performance Analysis

5.1 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 Subtree scheme, which is a representative solution to the broadcast encryption problem following the well-known Subset-Cover framework [33]. It employs a static logical key hierarchy, which is materialized with a full binary key tree of height h (equals to 3 in Figure 3), and thus can support up to 2^h ciphertext classes, a selected part of which is intended for an authorized delegatee.

In an ideal case as depicted in Figure 3(a), the delegatee can be granted the access to 2^{h_s} classes with the possession of only one key, where h_s is the height of a certain subtree (e.g., $h_s = 2$ in Figure 3(a)). On the other hand, to decrypt ciphertexts of a set of classes, sometimes the delegatee may have to hold a large number of keys, as depicted in Figure 3(b). Therefore, we are interested in n_a, the *number of symmetric-keys to be assigned* in this hierarchical key approach, in an average sense.

We assume that there are exactly 2^h 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_a should also be 0, which means no access to any of the classes; if $r = 100\%$, n_a 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 2^h classes. Consequently, one may expect that n_a may first increase with r, and may decrease later. We set $r = 10\%, 20\%, \cdots, 90\%$, and choose the portion in a random manner to model an arbitrary "delegation pattern" for different delegatees. For each combination of r and h, we randomly generate 10^4 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[6]. 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 (2^h) is large and the delegation ratio (r) is fixed, this kind of random delegation achieves roughly the same key

[6] Legend: h: The height of the binary tree: there are total 2^h ciphertext classes, n_a: The number of keys to be assigned, N: The total number of keys in the hierarchy, r: The delegation ratio: the ratio of the delegated ciphertext classes to the total classes.

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%

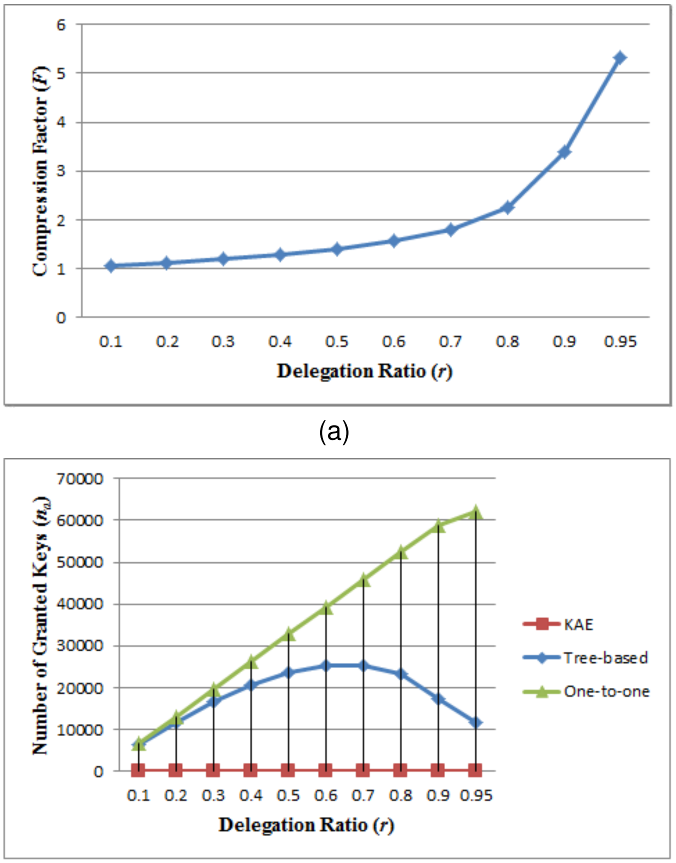
TABLE 2
Compression ratios for different delegation ratios and tree heights

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.

We then turn our focus to the *compression*[7] *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 ($r2^h$) 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.

A comparison of the number of granted keys between three methods is depicted in Figure 5(b). We can see that if we grant the key one by one, the number of granted keys would be equal to the number of the delegated ciphertext classes. With the tree-based structure, we can save a number of granted keys according to the delegation ratio. On the contrary, in our proposed approach, the delegation of decryption can be efficiently implemented with the aggregate key, which is only of fixed size.

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.



(a)

(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

5.2 Performance of Our Proposed Schemes

Our approaches allow the compression factor F ($F = n$ in our schemes) to be a tunable parameter, at the cost of $O(n)$-sized system parameter. Encryption can be done in constant time, while decryption can be done in $O(|\mathcal{S}|)$ group multiplications (or point addition on elliptic curves) with 2 pairing operations, where $\mathcal{S}$ is the set of ciphertext classes decryptable by the granted aggregate key and $|\mathcal{S}| \leq n$. As expected, key extraction requires $O(|\mathcal{S}|)$ 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) Library[8] version 0.4.18 for the underlying elliptic-curve group and pairing operations. Since the granted key can be as small as one $\mathbb{G}$ element, and the ciphertext only contains two

[7] As discussed, we are not proposing a compression mechanism, yet we effectively save the costly secure storage requirement.

[8] http://crypto.stanford.edu/pbc

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

TABLE 3
Performance of our basic construction for $h = 16$ with respect to different delegation ratio r (in milliseconds)

$\mathbb{G}$ and one $\mathbb{G}_T$ elements, we used (symmetric) pairings over Type-A (supersingular) curves as defined in the PBC library which offers the highest efficiency among all types of curves, even though Type-A curves do not provide the shortest representation for group elements. In our implementation, p is a 160-bit Solinas prime, which offers 1024-bit of discrete-logarithm security. With this Type-A curves setting in PBC, elements of groups $\mathbb{G}$ and $\mathbb{G}_T$ take 512 and 1024 bits to represent, respectively.

The test machine is a Sun UltraSparc IIIi system with dual CPU (1002 MHz) running Solaris, each with 2GB RAM. The timings reported below are averaged over 100 randomized runs. In our experiment, we take the *number of ciphertext classes* $n = 2^{16} = 65536$. The Setup algorithm, while outputting $(2n + 1)$ elements by doing $(2n - 2)$ exponentiations, can be made efficient by preprocessing function offered by PBC, which saves time for exponentiating the same element (g) in the long run. This is the only "low-level" optimization trick we have used. All other operations are implemented in a straightforward manner. In particular, we did not exploit the fact that $\hat{e}(g_1, g_n)$ will be exponentiated many times across different encryptions. However, we pre-computed its value in the setup stage, such that the encryption can be done without computing any pairing.

Our experiment results are shown in Table 3. The execution times of Setup, KeyGen, 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 $\mathcal{S}$). Our timing results also conform to what can be seen from the equation in Extract and Decrypt — two pairing operations take negligible time, the running time of Decrypt is roughly a double of Extract. Note that our experiments dealt with up to 65536 number of classes (which is also the compression factor), and should be large enough for fine-grained data sharing in most situations.

Finally, we remark that for applications where the number of ciphertext classes is large but the *non-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 $\mathbb{G}$. For $n = 2^{16}$, the system parameter requires approximately 2.6 megabytes, which is as large as a lower-quality MP3 file or a higher-resolution JPEG file that a typical cellphone can store more than a dozen of them. But we saved expensive *secure* storage without the hassle of managing a hierarchy of delegation classes.

6 New Patient-Controlled Encryption

Motivated by the nationwide effort to computerize America's medical records, the concept of patient-controlled encryption (PCE) has been studied [8]. 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, 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.

7 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 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 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 [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," 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.

[13] W.-G. Tzeng, "A Time-Bound Cryptographic Key Assignment Scheme for Access Control in a Hierarchy," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 14, no. 1, pp. 182–188, 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.

[16] Y. Sun and K. J. R. Liu, "Scalable Hierarchical Access Control in Secure Group Communications," in *Proceedings of the 23th IEEE International Conference on Computer Communications (INFOCOM '04)*. IEEE, 2004.

[17] 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.

[18] J. Benaloh, "Key Compression and Its Application to Digital Fingerprinting," Microsoft Research, Tech. Rep., 2009.

[19] B. Alomair and R. Poovendran, "Information Theoretically Secure Encryption with Almost Free Authentication," *J. UCS*, vol. 15, no. 15, pp. 2937–2956, 2009.

[20] 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.

[21] A. Sahai and B. Waters, "Fuzzy Identity-Based Encryption," in *Proceedings of Advances in Cryptology - EUROCRYPT '05*, ser. LNCS, vol. 3494. Springer, 2005, pp. 457–473.

[22] 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.

[23] F. Guo, Y. Mu, and Z. Chen, "Identity-Based Encryption: How to Decrypt Multiple Ciphertexts Using a Single Decryption Key," in *Proceedings of Pairing-Based Cryptography (Pairing '07)*, ser. LNCS, vol. 4575. Springer, 2007, pp. 392–406.

[24] M. Chase and S. S. M. Chow, "Improving Privacy and Security in Multi-Authority Attribute-Based Encryption," in *ACM Conference on Computer and Communications Security*, 2009, pp. 121–130.

[25] T. Okamoto and K. Takashima, "Achieving Short Ciphertexts or Short Secret-Keys for Adaptively Secure General Inner-Product Encryption," in *Cryptology and Network Security (CANS '11)*, 2011, pp. 138–159.

[26] R. Canetti and S. Hohenberger, "Chosen-Ciphertext Secure Proxy Re-Encryption," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. ACM, 2007, pp. 185–194.

[27] C.-K. Chu and W.-G. Tzeng, "Identity-Based Proxy Re-encryption Without Random Oracles," in *Information Security Conference (ISC '07)*, ser. LNCS, vol. 4779. Springer, 2007, pp. 189–202.

[28] C.-K. Chu, J. Weng, S. S. M. Chow, J. Zhou, and R. H. Deng, "Conditional Proxy Broadcast Re-Encryption," in *Australasian Conference on Information Security and Privacy (ACISP '09)*, ser. LNCS, vol. 5594. Springer, 2009, pp. 327–342.

[29] S. S. M. Chow, J. Weng, Y. Yang, and R. H. Deng, "Efficient Unidirectional Proxy Re-Encryption," in *Progress in Cryptology - AFRICACRYPT 2010*, ser. LNCS, vol. 6055. Springer, 2010, pp. 316–332.

[30] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 1–30, 2006.

[31] D. Boneh, C. Gentry, and B. Waters, "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys," in *Proceedings of Advances in Cryptology - CRYPTO '05*, ser. LNCS, vol. 3621. Springer, 2005, pp. 258–275.

[32] L. B. Oliveira, D. Aranha, E. Morais, F. Daguano, J. Lopez, and R. Dahab, "TinyTate: Computing the Tate Pairing in Resource-Constrained Sensor Nodes," in *Proceedings of 6th IEEE International Symposium on Network Computing and Applications (NCA '07)*. IEEE, 2007, pp. 318–323.

[33] D. Naor, M. Naor, and J. Lotspiech, "Revocation and Tracing Schemes for Stateless Receivers," in *Proceedings of Advances in Cryptology - CRYPTO '01*, ser. LNCS. Springer, 2001, pp. 41–62.

[34] 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.

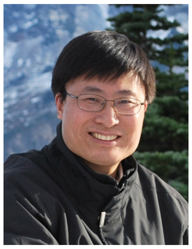
[35] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical Identity Based Encryption with Constant Size Ciphertext," in *Proceedings of Advances in Cryptology - EUROCRYPT '05*, ser. LNCS, vol. 3494. Springer, 2005, pp. 440–456.

[36] D. Boneh, R. Canetti, S. Halevi, and J. Katz, "Chosen-Ciphertext Security from Identity-Based Encryption," *SIAM Journal on Computing (SIAMCOMP)*, vol. 36, no. 5, pp. 1301–1328, 2007.

Cheng-Kang Chu received his Ph.D. in Computer Science from National Chiao Tung University (Hsinchu, Taiwan). After a postdoctoral fellowship in Singapore Management University with Prof. Robert H. Deng, he joined Cryptography and Security department at Institute for Infocomm Research (I2R) as a Research Scientist. He has had a long-term interest in the development of new technologies in Applied Cryptography, Cloud Computing Security, Wireless Sensor Network Security and Smart Grid Security. Now he is mainly working on a project to develop security techniques in large scale shared storage systems. He has published many research papers in major conferences like PKC, CT-RSA, ACNS, etc. and received the best student paper award in ISC 2007. He also served as vice chair of IEEE CCNC 2012 and on the program committee of many international conferences including TrustBus, WISTP, IEEE CCNC, IEEE CloudCom, etc.

Sherman S. M. Chow joined the Department of Information Engineering at the Chinese University of Hong Kong as an assistant professor in November 2012. He was a research fellow at Department of Combinatorics and Optimization, University of Waterloo, a position he commenced after receiving his Ph.D. degree from the Courant Institute of Mathematical Sciences, New York University. He interned at NTT Research and Development (Tokyo), Microsoft Research (Redmond) and Fuji Xerox Palo Alto Laboratory, and has made research visits to U. Maryland, U. Calgary, U. Texas, HKU, MIT, and Queensland University of Technology. These visits resulted in US patent applications and also in publications at major conferences such as ACM CCS and IACR EUROCRYPT. His research interests are applied cryptography, privacy and distributed systems security in general. He serves on the program committees of several international conferences including ASIACRYPT 2012-2013, ACNS 2012-2013, ASIACCS 2013, IEEE-CNS 2013 and Financial Crypt. 2013.

Wen-Guey Tzeng received his BS degree in Computer Science and Information Engineering from National Taiwan University, Taiwan, 1985; and MS and PhD degrees in Computer Science from the State University of New York at Stony Brook, USA, in 1987 and 1991, respectively. He joined the Department of Computer Science, National Chiao Tung University, Taiwan, in 1991. Professor Tzeng now serves as Chairman of the department. His current research interests include Cryptology, Information Security and Network Security.



Jianying Zhou is a senior scientist at Institute for Infocomm Research, and heads the Network Security Group. He received PhD in Information Security from University of London. His research interests are in computer and network security, mobile and wireless communications security, cloud security, and smart grid security.

Robert H. Deng has been a Professor at the School of Information Systems, Singapore Management University since 2004. Prior to this, he was Principal Scientist and Manager of Infocomm Security Department, Institute for Infocomm Research, Singapore. His research interests include data security and privacy, multimedia security, network and system security. He was the Associate Editor of the IEEE Transactions on Information Forensics and Security from 2009 to 2012. He is currently Associate Editor of IEEE Transactions on Dependable and Secure Computing, Associate Editor of Security and Communication Networks (John Wiley), and a member of Editorial Board of Journal of Computer Science and Technology (the Chinese Academy of Sciences). He is the co-chair of the Steering Committee of the ACM Symposium on Information, Computer and Communications Security (ASIACCS).

He received the University Outstanding Researcher Award from the National University of Singapore in 1999 and the Lee Kuan Yew Fellow for Research Excellence from the Singapore Management University in 2006. He was named Community Service Star and Showcased Senior Information Security Professional by $(ISC)^2$ under its Asia-Pacific Information Security Leadership Achievements program in 2010. He received the Distinguished Paper Award of the 19th Annual Network & Distributed System Security Symposium (NDSS 2012) and the Best Paper Award of the 13th Joint IFIP TC6 & TC11 Conference on Communications and Multimedia Security (CMS 2012).