Scalable Data Sharing Technique using Key-Aggregate Cryptosystem in Cloud Storage

Sudarshan S
M.Tech, CNE student
T.John Institute Of Technology
Bangalore, India

Srinivasa H P
Associate Professor, Dept. of CSE
T.John Institute Of Technology
Bangalore, India

***Abstract*— Recently Data sharing is very important in cloud storage. A new technique is proposed for efficient sharing of data with others in cloud storage in secured manner. We design an efficient public-key encryption scheme which fulfils flexible delegation of decryption rights by a constant-size decryption key of any subset of the ciphertexts. The basic idea is to aggregate a key by making as small single key from any set of secret keys. So, a constant-size aggregate key for flexible choices of cipher text set can be released by the secret key holder in cloud storage. The encrypted files outside the set remain confidential. This single compact aggregate key can be efficiently sent to others or be stored in a smart card with very limited secure storage. The proposed idea is the better public-key patient-controlled encryption for flexible hierarchy.**

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

I.INTRODUCTION

Recently Cloud storage is rapidly gaining popularity. In company settings, there is rise in demand for data outsourcing, which manages the enterprise data. Cloud storage is used as a back drop behind many online services. File sharing or remote access of data nowadays has become an easy task. With the help of present wireless technologies, data users can access all of their data and emails by a mobile phone from anywhere in the world.

To provide data privacy, a normal way is to depend on the server to provide the access rights after authentication which means any unexpected access privilege will expose the whole data. In a shared cloud computing environment, this becomes even worse. Data from different clients can be loaded on separate virtual machines that reside on a single physical machine. Data in a target VM could be stolen by instantiating another VM co-resident with the target one. To check the availability of files, there are few cryptographic schemes which allow a third-party to check the existence of files without leaking or compromising any of the data. Cloud users does not hold the belief that the cloud server provides confidentiality The cloud 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. Like, bloggers can allow their friends to view a subset of their personal pictures; a company may grant her employees access to a particular portion of important data. The main problem here is how to share encrypted data effectively. Users can also download the encrypted data from the storage, decrypt them, then send them to others for sharing the files, but the value of cloud storage loses. Data Users should be able to control the access rights of the data sharing to other users by which they can access data from the server directly. Finding an secure and efficient way to share data in cloud storage is not an easy task. Let's take Dropbox as an example for illustration.

Assume Data owner puts all their personal photos on Dropbox, and she does not want to expose her photos to everyone. Data owner wont believe the protection provided by the dropbox, so owner encrypts all the photos with own keys before uploading to cloud. If data owner friends asks to share the photos, data owner can then use the share function of Dropbox, but the problem is how to delegate the decryption rights for the particular photos to other users. A possible option Data owner can choose is to securely send end users the secret keys needed. There are two possible ways for her under the traditional encryption paradigm:

- Data owner encrypts all files with a single encryption key and gives end user the secret key directly.
- Data owner encrypts files with different keys and sends end users the corresponding secret keys.

The first method is not adequate as all data which is not needed may be also leaked to end user. In the second method, practical concerns on efficiency exist. The number keys is as many as the total number of the photos shared. Transferring these many secret keys requires a secure channel, and storing of these keys requires expensive storage.

The better solution for the above existing problem is that Data owner encrypts all files with distinct public-keys, but sends end user a single- constant-size decryption key. The decryption key is sent via a secure channel and kept secret, small key size is always desirable. These secret keys can be stored in the tamper-proof memory. The researchers mainly focus on minimizing the communication requirements like aggregate signature, However, not much has been done about the key itself.

1.1 Our Contributions

A fundamental problem we often face in modern cryptography is about deligate the secrecy of knowledge to perform cryptographic functions such as encryption and authentication multiple times. Here lets study how to make a decryption key more powerful so that it allows decryption of multiple ciphertexts easily, without increasing the size. Specifically, our problem statement is –

"Designing a efficient public-key encryption scheme that supports flexible delegation that allows any subset of the ciphertexts are decryptable by a constant-size decryption key."

This problem is solved by introducing a type of public-key encryption called key-aggregate cryptosystem (KAC). In KAC, users encrypt a message under a public-key and 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 is used to extract secret keys for all classes. The extracted key can be an aggregate key which is as small as a secret key for a single class, but aggregates the power of many keys, that is, the decryption power for any subset of ciphertext classes.

With this solution, Data owner can simply send end users a single aggregate key thorugh a secure e-mail. End users can download the encrypted photos from Data owner's Dropbox space and then decrypt these encrypted photos with the help of aggregate key . The scenario is depicted in Figure 1.

The sizes of all the key and aggregate key in this KAC method 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.

Existing results achieve a similar property with a constant-size decryption key, but the classes need to conform to pre-defined hierarchical relationship. KAC is flexible so that this constraint is eliminated, that is, no relation is required between the classes.

II.KEY-AGGREGATE ENCRYPTION

The framework and definition for keyaggregate encryption is described above. Then now describing how to use KAC in a scenario of its applications in cloud storage.

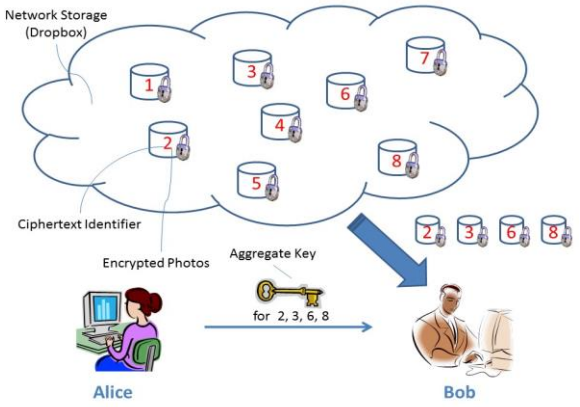


Fig. 1. Alice shares files with identifiers 2, 3, 6 and 8 with Bob by sending him a single aggregate key.

2.1 *Framework*

A key-aggregate encryption scheme(KAC) consists of five polynomial-time algorithms as described below.

The data owner establishes the public system parameter through Setup and generates a public/master-secret key pair via KeyGen. Messages can be encrypted via Encrypt by the one who also decides what ciphertext class is associated with the plaintext message to be encrypted. The data owner uses the master-secret to generate an aggregate constant size decryption key for a set of ciphertext classes via Extract. The generated keys can be passed to delegatees securely. Any user with an aggregate key can be able to decrypt any ciphertext provided that the ciphertext's class is contained in the aggregate key via Decrypt.

- Setup(1^{λ},n): executed by the data owner to setup an account on an untrusted server. On input a security level parameter 1^{λ} and the number of ciphertext classes n (i.e., class index should be an integer bounded by 1 and n), it outputs the public system parameter param, which is omitted from the input of the other algorithms for brevity.
- KeyGen: executed by the data owner to 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 mastersecret key msk and a set S of indices corresponding to different classes, it outputs the aggregate key for set S denoted by K_S.
- Decrypt(K_S,S,i,C): executed by a delegatee who received an aggregate key K_S generated by Extract. On input K_S, the set S, an index i denoting the

There are two functional requirements:

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

$$\Pr[\text{Decrypt}(K_S,S,i,C) = m : \text{param} \leftarrow \text{Setup}(1^{\lambda},n),$$
$$(\text{pk},\text{msk}) \leftarrow \text{KeyGen}(),C \leftarrow \text{Encrypt}(\text{pk},i,m),$$
$$K_S \leftarrow \text{Extract}(\text{msk},S)] = 1.$$

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

2.2 *Sharing Encrypted Data*

The main application of Key Aggregate Cryptosystem is data sharing. The key aggregation property is useful when we need the delegation to be efficient and flexible. This schemes enable a content provider to share data in a confidential and selective and secure way, with a fixed and small ciphertext

expansion, by distributing to each authorized user a single and small aggregate key.

The main idea of data sharing in cloud storage using KAC is described here, illustrated in Figure 2. If Data owner wants to share data $m_1,m_2,...,m_v$ on the server. Owner initialy performs Setup(1^λ,n) to get param and execute KeyGen to generate the public/master-secret key pair (pk,msk). The system parameter param and public-key pk are made public but master-secret key msk is kept secret by Data owner. Data owner can encrypt every m_i by C_i = Encrypt(pk,i,m_i). Then the encrypted data are uploaded to the cloud server.

With param and pk, people who cooperate with Data owner can update owner's data on the server. Once data owner is willing to share a set S of data with a friend, they can compute the aggregate key K_S for End user by performing Extract(msk,S). K_S is just a constant size key, it is easy to be sent to end user through a secure e-mail. After getting the aggregate key, end users can download the data that he is authorized to access. That is, for each $i \in S$, end users downloads C_i from the server. With the help of aggregate key K_S, Bob can decrypt each C_i by Decrypt(K_S,S,i,C_i) for each $i \in S$.

III.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 aim to minimize the expense in storing and managing

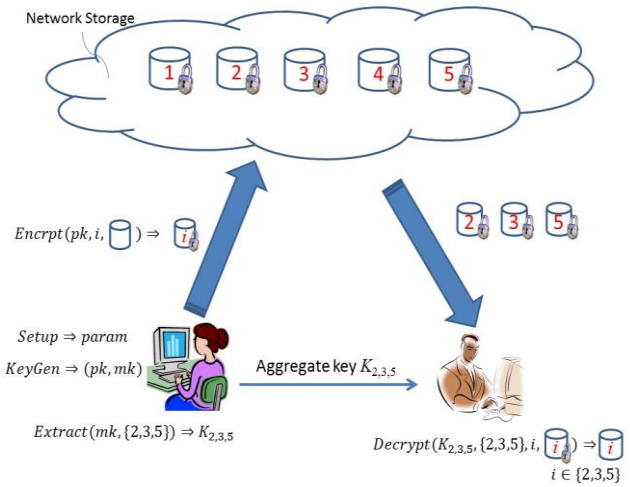


Fig. 2. Using KAC for data sharing in cloud storage

secret keys for general cryptographic use. Utilizing a tree structure, a key for a given branch can be used to derive the keys of its descendant nodes (but not the other way round). Just granting the parent key implicitly grants all the keys of its descendant nodes. Sandhu proposed a method to generate a tree hierarchy of symmetric keys by using repeated evaluations of pseudorandom function/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. 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.

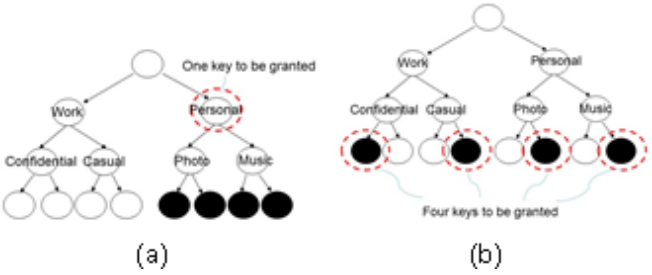


Fig. 3. Compact key is not always possible for a fixed Hierarchy

	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

In Figure 3(a), if Data owner wants to share the personal files, owner only needs to provide the key for the node private, which automatically provides the delegation of the keys of all the descendant nodes ("photo", "music"). This is the ideal for case where classes shared belong to the same branch and thus a parent key of them is sufficient.

It is still difficult to achieve for general cases. As shown in Figure 3(b), if owner shares his data at work with a colleague who has the permission to see some of the private data, what owner can do is to give more keys, which increases the total key size. This approach is not flexible when the classifications are more complex and owner wants to share different sets of files to different users. For this delegate 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 simultaneously.

3.2 *Compact Key in Symmetric-Key Encryption*

Motivated by the same problem of supporting flexible hierarchy in decryption power delegation, Benaloh *et al.* presented an encryption scheme which is originally proposed for concisely transmitting large number of keys in broadcast scenario. 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 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^*_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 S^0 can be generated (with the knowledge of $\varphi(N)$) as $k_{S^0} = Y^{1/\prod_{j \in S^0}(e_j)} \bmod N$. For those who have been delegated the access rights for S where $S^0 \subset S$, k_{S^0} can be computed by $k_S^{\prod_{j \in S \setminus S^0}(e_j)}$. 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 encrypter 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. Sharing of decryption power is not a concern in these schemes.

3.3 *Compact Key in Identity-Based Encryption*

Identity-based encryption (IBE) 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. 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.* tried to build IBE with key aggregation. One of their schemes assumes random oracles but another 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, 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, 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) 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 ∨ 3 ∨ 6 ∨ 8), anyone can decrypt ciphertext with class 2,3,6 or 8. The major concern in ABE is collusion-resistance. The size of the key often increases with the number of attributes it encompasses.

Proxy re-encryption(PRE) is used to delegate the decryption rights of some ciphertexts without sending the secret key to the delegate. A PRE method allows Data owner to delegate to the server, the ability to convert the ciphertexts encrypted under public-key into ones for End user. PRE is known to have many applications which include cryptographic file system. Data owner has to trust the proxy that it only converts ciphertexts according to their instruction, which we need to avoid first. If the proxy collides with end user, some form of Data owner secret key can be obtained which can decrypt owner's ciphertexts without the help of end user. The transformation key of server need to be protected. Using PRE moves the secure key storage requirement from the delegate to the proxy server. It is undesirable to let the proxy reside in the storage server. That will also be inconvenient since every decryption need separate interaction with the proxy.

IV.CONCRETE CONSTRUCTIONS OF KAC

Let G and G_T be two cyclic groups of prime order p and $e\hat{}$: $G\times G\rightarrow G_T$ 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 $g \in G$, $e\hat{}(g,g) \neq 1$.

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 this scheme is inspired from the collusion-resistant broadcast encryption scheme which is proposed by Boneh *et al.*. Their scheme supports constant-size secret keys, though every key only has the power for decrypting ciphertexts of a particular index. Thus there is need to devise a new Extract algorithm and the corresponding Decrypt algorithm.

- Setup(1^λ,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 $\alpha \in_R Z_p$. Compute $g_i = g^\alpha \in G$ for $i = 1,\cdots,n,n+2,\cdots,2n$. Output the system parameter as 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 $\gamma \in_R Z_p$, output the public and master-secret key pair: (pk = $v = g^\gamma$,msk = γ).

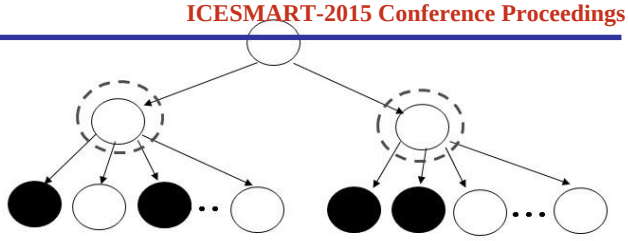


Fig 4: key assignment in KAC approach

- Encrypt(pk,i,m): For a message $m \in G_T$ and an index $i \in \{1,2,\cdots,n\}$, randomly pick $t \in_R Z_p$ and compute the ciphertext as $C = \langle g^t, (vg_i)^t, m \cdot e\hat{}(g_1,g_n)^t\rangle$.
- Extract(msk = γ,S): For the set S of indices j's, the aggregate key is computed as $K_S = \prod_{j\in S} g_{n+1-j}^\gamma$.
  Since S does not include 0, $g_{n+1-j} = g^{\alpha_{n+1-j}}$ can always be retrieved from param.
- Decrypt(K_S,S,i,$C = \langle c_1,c_2,c_3\rangle$): If $i \notin$ S, output $\perp$. Otherwise, return the message: $m = c_3 \cdot e\hat{}(K_S \cdot \prod_{j\in S, j\neq i} g_{n+1-j+i}, c_1)/\hat{e}(\prod_{j\in S} g_{n+1-j}, c_2)$.

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

For correctness, we can see that

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

4.1.1 Performance

For encryption, the value $e\hat{}(g_1,g_n)$ 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.

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 Fig. 4. Key assignment in our approach 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. The system parameter can also be generated by a trusted party, shared between all users and even hardcoded 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 $(pk_2,msk_2),\cdots,(pk_\cdot,msk_\cdot)$. Each class now is indexed by a 2-level index in $\{(i,j)|1 \le i \le \grave{},1 \le j \le n\}$ 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. 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 the details on how encryption and decryption work when√ the public-key is extended, which is similar to the "n-approach".

- Setup and KeyGen: Same as the basic construction.
- Extend(pk_l,msk_l): Execute KeyGen() to get $(v_{l+1},\gamma_{l+1}) \in G \times Z_p$, output the extended public and master-secret keys as
  $pk_{l+1} = (pk_l,v_{l+1}),msk_{l+1} = (msk_l,\gamma_{l+1})$.
- Encrypt(pk_l,(a,b),m): Let $pk_l = \{v_1,\cdots,v_l\}$. For an index $(a,b),1 \le a \le l,1 \le b \le n$, pick $t \in_R Z_p$, output the ciphertext as $C = h g^t,(v_a g_b)^t, m \cdot e\hat{}(g_1,g_n)^t i$.
- Extract(msk_l,S_l): Let $msk_l = \{\gamma_1,\gamma_2,\cdots,\gamma_l\}$. For a set S_l of indices $(i,j),1 \le i \le l,1 \le j \le n$, get $g_{n+1-j} = g^{\alpha_{n+1-j}}$ from param, output:

$$K_{S_l} = \Big(\prod_{(1,j)\in S_l} g^{\gamma_1}_{n+1-j}, \prod_{(2,j)\in S_l} g^{\gamma_2}_{n+1-j}, \cdots, \prod_{(l,j)\in S_l} g^{\gamma_l}_{n+1-j} \Big).$$

- Decrypt(K_{S_l},S_l,(a,b),C): If $(a,b) \notin S_l$, output $\perp$. Otherwise, let $K_{S_l} = (d_1,\cdots,d_l)$ and $C = h c_1,c_2,c_3 i$. Output the message:

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

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:

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

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, pk_1 and 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 pk_1 with a new key pk_2. The small aggregate key size minimizes the communication overhead for transferring the new key.

V.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.

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 ~ 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 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.

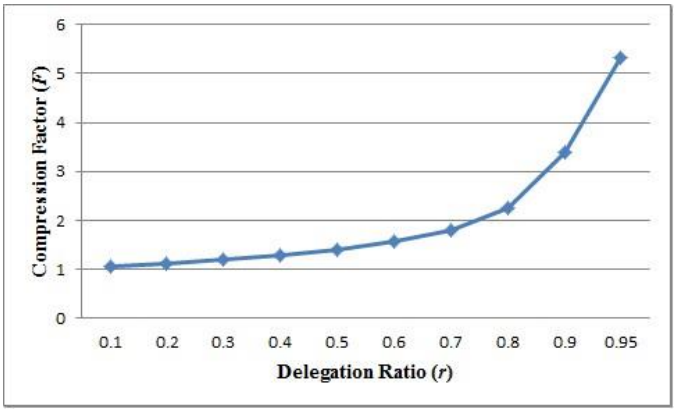
In our experiment, the delegation is randomly chosen. It models the situation that the needs for delegating to different users may not be predictable as time goes by, even after a careful initial planning. This gives empirical evidences to support our thesis that hierarchical key assignment does not save much in all cases.

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 %

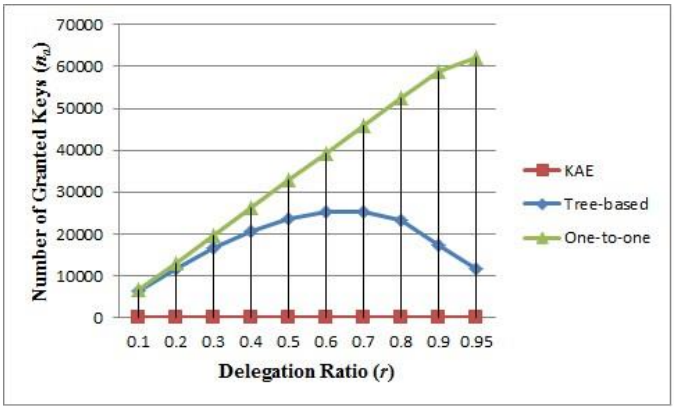
TABLE 2
Compression ratios for different delegation ratios and tree heights

5.2 Performance of Our Proposed Schemes

This approaches allow the compression factor F 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 $O(|S|)$ group multiplications (or point addition on elliptic curves) with 2 pairing operations, where S is the set of ciphertext classes decryptable by the granted aggregate key and $|S| \leq n$. As expected, key extraction requires $O(|S|)$ group multiplications as well, which seems unavoidable.



(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

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 G element, and the ciphertext only contains two

G and one G_T elements, used (symmetric) pairings over Type-A 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 G and 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 this experiment, 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 $e\hat{}(g_1,g_n)$ 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 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 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 saved expensive secure storage without the hassle of managing a hierarchy of delegation classes.

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)

VI.NEW PATIENT-CONTROLLED ENCRYPTION

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.*, three solutions have been provided, which are symmetric-key PCE for fixed hierarchy, public-key PCE for fixed hierarchy, and RSA-based symmetric-key PCE for "flexible hierarchy" (which is the "set membership" access policy as we explained).

This 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.

VII.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 is the predefined bound of the number of maximum ciphertext classes. In cloud storage, the number of ciphertexts usually grows rapidly. So 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 leakageresilient cryptosystem 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, "PrivacyPreserving 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 ResourceConstrained 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, "IdentityBased 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*