KCSC Key Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage

K.Lavanya[1], V.V.A.S.Lakshmi[2], G.Sai Ganesh[3]

[1]*M.tech student, Narasaraopeta Institute of Technology, Narasaraopeta Guntur dist, A.P, india*
[2]*Associate Professor, Narasaraopeta Institute of Technology, Narasaraopeta Guntur dist, A.P, india*
[3]*Assistant Professor, Narasaraopeta Engineering college, Narasaraopeta, Guntur dist, A.P, india*
[1] Sailavanya45@gmail.com

***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.**

I. INTRODUCTION

Cloud storage is gaining popularity recently. In en-terprise 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 en-sure 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 re-lied on number-theoretic assumptions is more desirable, whenever the user is not perfectly happy with trusting the security of the VM or the honesty of the technical staff. These users are motivated to encrypt their data with their own keys before uploading them to the server.

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 protec-tion mechanisms provided by Dropbox, so she encrypts all the photos using her own keys before uploading. One day, Alice's friend, Bob, asks her to share the photos taken over all these years which Bob appeared in. Alice can then use the share function of Dropbox, but the problem now is how to delegate the decryption rights for these photos to Bob. A possible option Alice can choose is to securely send Bob the secret keys involved. Naturally, there are two extreme ways for her under the traditional encryption paradigm:

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

Alice encrypts files with distinct keys and sends Bob the corresponding secret keys.

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

Encryption keys also come with two flavors — sym-metric 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 up-load 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 de-sirable. 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 crypto-graphic functions (e.g. encryption, authentication) mul-tiple 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

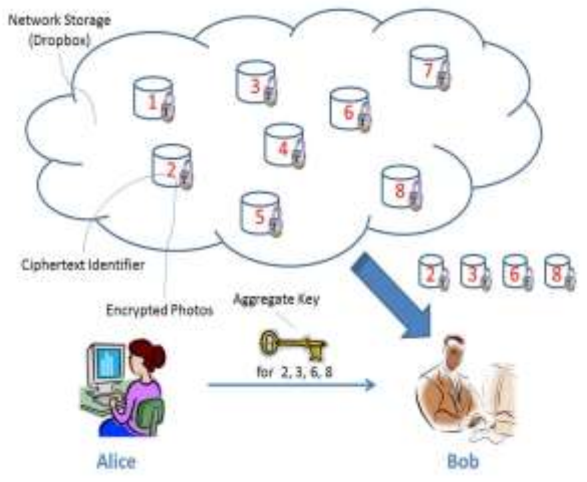


Fig. 1. Alice shares files with identifiers 2, 3, 6 and 8 with Bob by sending him a single aggregate key.

"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 featur-ing 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 dif-ferent 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.

II. *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 param-eter 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].

Setup(1 ; n): executed by the data owner to setup an account on an untrusted server. On input a security level parameter 1 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 del-egating the decrypting power for a certain set of ci-phertext classes to a delegatee. On input the master-secret key msk and a set S of indices corresponding to different classes, it outputs the aggregate key for set S denoted by 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 ciphertext class the ciphertext C belongs to, and C, it outputs the decrypted result m if i 2 S.

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; : : : ; m on the server. She first performs Setup(1 ; n) to get param and execute KeyGen to get the public/master-secret key pair (pk; msk). The system parameter param and public-key pk can be made public and master-secret key msk should be kept secret by Alice. Anyone (including Alice herself) can then encrypt each m_i by C_i = Encrypt(pk; i; m_i). The encrypted data are uploaded to the server.

With param and pk, people who cooperate with Alice can update Alice's data on the server. Once Alice is willing to share a set S of her data with a friend Bob, she can compute the aggregate key K_S for Bob by performing Extract(msk; S). Since K_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 2 S, Bob downloads C_i (and some needed values in param) from the server. With the aggregate key K_S, Bob can decrypt each C_i by Decrypt(K_S; S; i; C_i) for each i 2 S. minimize the expense in storing and managing secret keys for general cryptographic use. Utilizing a tree struc-ture, 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.

Most of these schemes pro-duce 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.

III. COMPACT KEY IN SYMMETRIC-KEY ENCRYPTION

Motivated by the same problem of supporting flexi-ble hierarchy in decryption power delegation (but in symmetric-key setting), Benaloh et al. [8] presented an encryption scheme which is originally proposed for con-cisely 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 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.

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.

3.1 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 (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.

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 imprac-tical in many situations such as shared cloud storage.

3.2 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 _ 3 _ 6 _ 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 includ-ing 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.

IV. CONCRETE CONSTRUCTIONS OF KAC

4.1 A Basic Construction

The design of our basic scheme is inspired from the collusion-resistant broadcast encryption scheme pro-posed by Boneh et al. [31]. Although their scheme supports constant-size secret keys, every key only has
the power for decrypting ciphertexts associated to a particular index. We thus need to devise a new Extract algorithm and the corresponding Decrypt algorithm.

Setup(1 ; n): Randomly pick a bilinear group G of prime order p where 2 p 2 $^{+1}$, a generator g 2 G and 2$_R$ Z$_p$. Compute g$_i$ = g ^{i} 2 G for i = 1; ; n; n+2; ; 2n. Output the system parameter as param = hg; g$_1$; ; g$_n$; g$_{n+2}$; ; g$_{2n}$i (can be safely deleted after Setup).
Note that each ciphertext class is represented by an index in the integer set f1; 2; ; ng, where n is the maximum number of ciphertext classes.
KeyGen(): Pick 2$_R$ Z$_p$, output the public and master-secret key pair: (pk = v = g ; msk =).
Encrypt(pk; i; m): For a message m 2 G$_T$ and an index i 2 f1; 2; ; ng, randomly pick t 2$_R$ Z$_p$ and compute the ciphertext as C = hg^t; (vg$_i$)^t; m e^(g$_1$; g$_n$)^{t}i.
Extract(msk = ; S): For the set S of indices j's, the aggregate key is computed as K$_S$ = ^{Q} g$_{n+1\ j}$.

Decrypt(K$_S$; S; i; C = hc$_1$; c$_2$; c$_3$i): If i 2= S, output ?. Otherwise, return the message:

V. PERFORMANCE ANALYSIS

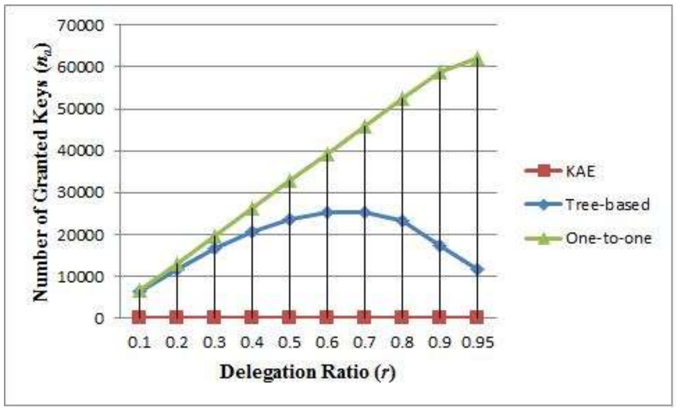
5.1 Compression Factors

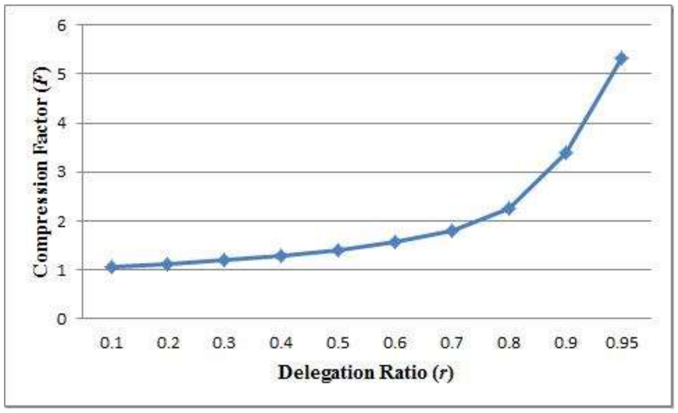
For a concrete comparison, we investigate the space requirements of the tree-based key assignment approach we described in Section 3.1. This is used in the Complete Subtree scheme, which is a representative solution to the broadcast encryption problem following the well-known Subset-Cover framework [33]. 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 dele-gatee can be granted the access to 2^{hs} 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%; ; 90%, and choose the portion in a random manner to model an arbitrary "delegation pat-tern" 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 dele-gation 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 w hen 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.

VI. CONCLUSION AND FUTURE WORK

How to protect users' data privacy is a central ques-tion 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 Envi-ronment," 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 Cryptog-raphy and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th

[6] Birthday, ser. LNCS, vol. 6805. Springer, 2012, pp. 442–464.

[7] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, “Aggregate and Verifiably Encrypted Signatures from Bilinear Maps,” in

[8] Proceedings of Advances in Cryptology - EUROCRYPT ’03, ser. LNCS, vol. 2656. Springer, 2003, pp. 416–432.

[9] 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.

[10] 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.

[11] F. Guo, Y. Mu, Z. Chen, and L. Xu, “Multi-Identity Single-Key Decryption without Random Oracles,” in Proceedings of Informa-tion Security and Cryptology (Inscrypt ’07), ser. LNCS, vol. 4990. Springer, 2007, pp. 384–398.

[12] 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 Com-munications Security (CCS ’06). ACM, 2006, pp. 89–98.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] R. S. Sandhu, “Cryptographic Implementation of a Tree Hierarchy for Access Control,” Information Processing Letters, vol. 27, no. 2, pp. 95–98, 1988.

[18] 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.

[19] 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.

[20] J. Benaloh, “Key Compression and Its Application to Digital Fingerprinting,” Microsoft Research, Tech. Rep., 2009.

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

[22] 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.

[23] A. Sahai and B. Waters, “Fuzzy Identity-Based Encryption,” in

[24] Proceedings of Advances in Cryptology - EUROCRYPT ’05, ser. LNCS, vol. 3494. Springer, 2005, pp. 457–473.

Selected Paper from International Conference on Computing (NECICC-2k15)