International Journal of Advanced Technology and Innovative Research

www.ijatir.org

ISSN 2348–2370
Vol.08,Issue.19,
November-2016,
Pages:3631-3635

Key-Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage

KOMIRINENI SRUTHI[1], K. RAJENDAR[2]

[1]PG Scholar, Sindhura College of Engineering and Technology, Godavarikhani, TS, India, E-mail: sruthirao.k3@gmail.com.
[2]Asst Prof, Sindhura College of Engineering and Technology, Godavarikhani, TS, India, E-mail: kalavala.rajendar@gmail.com.

Abstract: Cloud storage is a storage of data online in cloud which is accessible from multiple and connected resources. Cloud storage can provide good accessibility and reliability, strong protection, disaster recovery, and lowest cost cloud storage having important functionality i.e. securely, efficiently, flexibly sharing data with others new public key encryption which is called as Key-aggregate cryptosystem (KAC). Key-aggregate cryptosystem produce constant size ciphertexts such that efficient delegation of decryption rights for any set of ciphertext are possible. Any set of secret keys can be aggregated and make them as single key, which encompasses 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. The difference is one can collect a set of secret keys and make them as small size as a single key with holding the same ability of all the keys that are formed in a group. This compact aggregate key can be efficiently sent to others or to be stored in a smart card with little secure storage.

Keywords: Random Oracles (RO), Transformation Key (TK), Key-Aggregate Cryptosystem (KAC).

I. INTRODUCTION

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. 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 encrypt or 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.

A. Problem Definition

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 cipher texts, without increasing its size. Specifically, our problem statement is "To design an efficient public-key encryption scheme which supports flexible delegation in the sense that any subset of the cipher texts (produced by the encryption scheme) is decrypt able by a constant-size decryption key (generated by the owner of the master-secret key)." We solve this problem by introducing a special type of public-key encryption which we call key-aggregate cryptosystem (KAC). 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 Drop box space and then use this aggregate key to decrypt these encrypted photos. The sizes of ciphertext, public-key, and 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 predefined hierarchical relationship. Our work is flexible in the sense that this constraint is eliminated, that is, no special relation is required between the classes.

II. EXISTING AND PROPOSED SYSTEMS

A. Existing System

There exist several expressive ABE schemes where the decryption algorithm only requires a constant number of pairing computations. Recently, Green et al. proposed a

remedy to this problem by introducing the notion of ABE with outsourced decryption, which largely eliminates the decryption overhead for users. Based on the existing ABE schemes, Green et al. also presented concrete ABE schemes with outsourced decryption. In these existing schemes, a user provides an untrusted server, say a proxy operated by a cloud service provider, with a transformation key TK that allows the latter to translate any ABE ciphertext CT satisfied by that user's attributes or access policy into a simple ciphertext CT', and it only incurs a small overhead for the user to recover the plaintext from the transformed ciphertext CT'. The security property of the ABE scheme with outsourced decryption guarantees that an adversary (including the malicious cloud server) be not able to learn anything about the encrypted message; however, the scheme provides no guarantee on the correctness of the transformation done by the cloud server. In the cloud computing setting, cloud service providers may have strong financial incentives to return incorrect answers, if such answers require less work and are unlikely to be detected by users.

B. Proposed System

We considered the verifiability of the cloud's transformation and provided a method to check the correctness of the transformation. However, the we did not formally define verifiability. But it is not feasible to construct ABE schemes with verifiable outsourced decryption following the model defined in the existing. Moreover, the method proposed in existing relies on random oracles (RO). Unfortunately, the RO model is heuristic, and a proof of security in the RO model does not directly imply anything about the security of an ABE scheme in the real world. It is well known that there exist cryptographic schemes which are secure in the RO model but are inherently insecure when the RO is instantiated with any real hash function. In this thesis work, firstly modify the original model of ABE with outsourced decryption in the existing to allow for verifiability of the transformations. After describing the formal definition of verifiability, we propose a new ABE model and based on this new model construct a concrete ABE scheme with verifiable outsourced decryption. Our scheme does not rely on random oracles. In this paper we only focus on CP-ABE with verifiable outsourced decryption. The same approach applies to KP-ABE with verifiable outsourced decryption. To assess the performance of our ABE scheme with verifiable outsourced decryption, we implement the CP-ABE scheme with verifiable outsourced decryption and conduct experiments on both an ARM-based mobile device and an Intel-core personal computer to model a mobile user and a proxy, respectively.

III. CONTEXT DIAGRAM OF PROJECT

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 as shown in Fig.1. 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. A key-aggregate encryption scheme consists of five polynomial-time algorithms as follows. The data owner establishes the public system parameter via Setup and generates a public/master-secret3 key pair via Key Gen. 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.

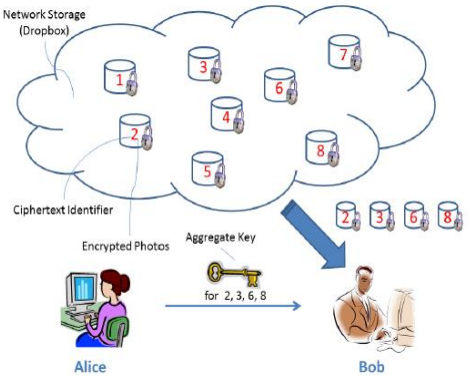


Fig.1. Context Diagram of key aggregation.

The generated keys can be passed to delegates securely (via secure e-mails or secure devices) finally; any user with an aggregate key can decrypt any ciphertext provided that the cipher text's class is contained in the aggregate key via decrypt. We call this as master-secret key to avoid confusion with the delegated key we will explain later. 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.

A. Algorithm and Modules

1. Secured Hashing Algorithm

There are several similarities in the evolution of hash function and that of symmetric block ciphers. We have seen that the increasing power of brute-force attacks and advances in cryptanalysis have led to the decline in the popularity of DES and in the design of newer algorithm with longer key lengths and with features designed to resist specific cryptanalytic attacks as shown in Fig.2. Similarly, advances in computing power and hash function cryptanalysis have led to the decline in the popularity of first MD4 and then MD5, two very popular hash functions. In response, newer hash algorithm have been developed with longer hash code length and with features designed to resist specific cryptanalytic attacks. Another point of similarity is the reluctance to depart from a proven structure. DES is based on the Feistel cipher, which in turn is based on the Substitution-permutation network proposal of Shannon. Many important subsequent block ciphers follow the feistel design because the design can be adapted to resist newly discovered cryptanalytic threats. If, instead, an entirely new design were used for a symmetric block cipher, there would be concern that the

structure itself opened up new avenues of attack not yet thought of. Similarly, most important modern hash functions follow the basic structure. This has proved to be a fundamentally sound structure and newer designs simply refine the structure and add to the hash code length. MD5, SHA-1, and RIPEMD-160. We then look at an internet-standard message authentication code. A hash function H is a transformation that takes a variable-size input m and returns a fixed-size string, which is called the hash value h (that is, h = H(m)). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography the hash functions are usually chosen to have some additional properties.

B. Modules

1. Setup Phase
2. Encrypt Phase
3. Key Gen Phase,
4. Decrypt Phase

Setup Phase: The setup algorithm takes no input other than the implicit security parameter. It outputs the public parameters PK and a master key MK.

Encrypt Phase: Encrypt (PK, M, A). The encryption algorithm takes as input the public parameters PK, a message M, and an access structure A over the universe of attributes. The algorithm will encrypt M and produce a ciphertext CT such that only a user that possesses a set of attributes that satisfies the access structure will be able to decrypt the message. We will assume that the ciphertext implicitly contains A.

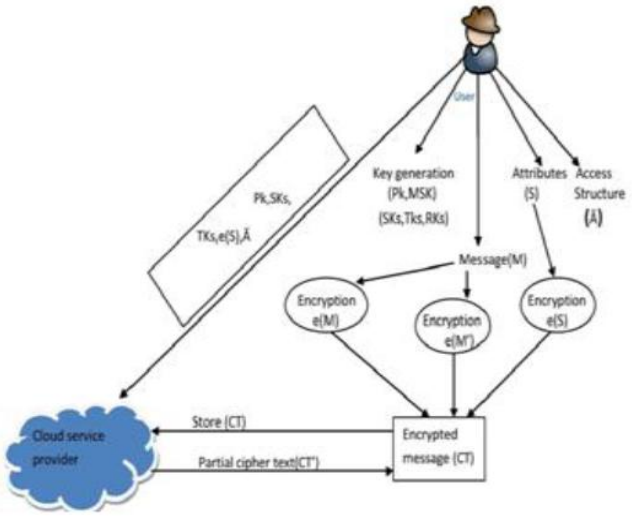


Fig.2. System architecture

Key Gen Phase: Key Generation (MK,S). The key generation algorithm takes as input the master key MK and a set of attributes S that describe the key. It outputs a private key SK.

Decrypt Phase: Decrypt (PK, CT, SK). The decryption algorithm takes as input the public parameters PK, a ciphertext CT, which contains an access policy A, and a private key SK, which is a private key for a set S of attributes. If the set S of attributes satisfies the access structure A then the algorithm will decrypt the ciphertext and return a message M.

IV. PERFORMANCE ANALYSIS

A. Compression Factors

For a concrete comparison, we investigate the space requirements of the tree-based key assignment approach we described. This is used in the Complete Sub tree scheme, which is a representative solution to the broadcast encryption problem following the well-knownSubset-Cover framework. It employs a static logical key hierarchy, which is materialized with a full binary key tree of height h 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, the delegatee 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 sub tree (e.g., $h_s = 2$). 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. 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 pattern" for different delegatees. For each combination of r and h, we randomly generate 104 different combinations of classes to be delegated, and the output key set size n_a is the average over random delegations. We tabulate the results in Table 1, where h = 16, 18, 20 respectively6. 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), 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.

TABLE 1. Compression ratios for different delegation ratios and tree heights

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%

We then turn our focus to the compression 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. Fig.3 (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 Fig. 3(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.

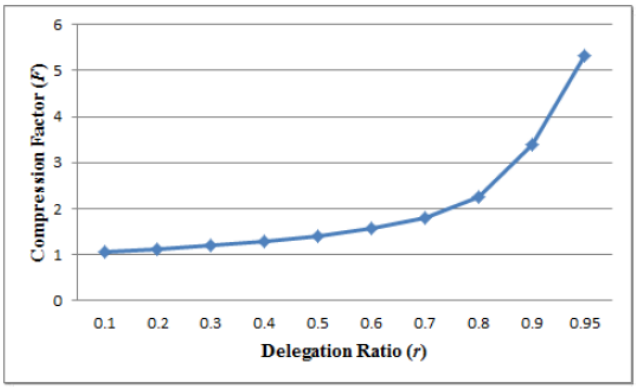
B. Performance of Our Proposed Schemes

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 $O(|S|)$ group multiplications (or point addition on elliptic curves) with 2 pairing operations, where S is the set of ciphertext classes decrypt able by the granted aggregate key and $|S| \leq n$. As expected, key extraction requires O ($|S|$) 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 version 0.4.18 for the underlying elliptic-curve group and pairing operations.

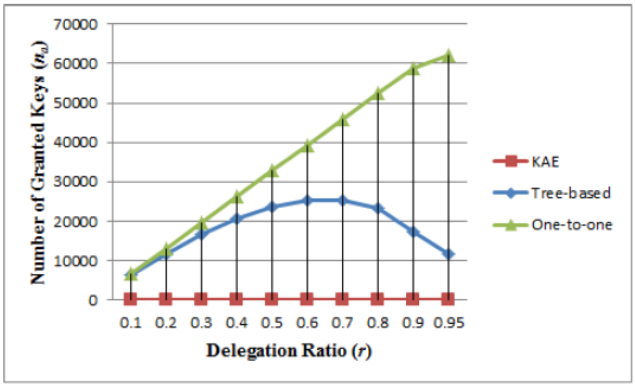
TABLE 2.Performance of our basic construction for h = 16 with respect to different delegation ratio r (in milliseconds)

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

Since the granted key can be as small as one G element, and the ciphertext only contains two G and one G_T elements, we used (symmetric) pairings over Type-A (super singular) curves as defined in the PBC library which offers the highest efficiency among all types of curves, even though Type-A curves do not provide the shortest representation for group elements.



(a)



(b)

Fig.3. (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

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 Ultra Sparc 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 = 216 = 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 2. The execution times of Setup, Key Gen, and Encrypt are independent of the delegation ratio r.

In our experiments, Key Gen 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 cell phone can store more than a dozen of them. But we saved expensive secure storage without the hassle of managing a hierarchy of delegation classes.

V. CONCLUSION

How to protect users' data privacy is a central question of cloud storage. With more mathematical tools, cryptographic schemes are getting more versatile and often involve multiple keys for a single application. In this article, we consider how to "compress" secret keys in public-key cryptosystems which support delegation of secret keys for different ciphertext classes in cloud storage. No matter which one among the power set of classes, the delegate 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.

Future Enhancement: In Future it can be upgraded with verifiable and recoverable. The ABE with verifiable provides us to verify the data whether it is modified or not. In future it can be upgraded by using Hash Chains such that we can identify the exact modified block and recover the remaining part of the data.

VI. REFERENCES

[1] Cheng-Kang Chu, Sherman S. M. Chow, Wen-Guey Tzeng, Jianying Zhou, and Robert H. Deng, Senior Member, IEEE, "Key-Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage", IEEE Transactions on Parallel and Distributed Systems, Vol. 25, No. 2, February 2014.

[2] 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.

[3] L. Hardesty, "Secure computers aren't so secure," MIT press, 2009, http://www.physorg.com/news176107396.html.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.