Symmetric Key Based Secure Resource Sharing

Bruhadeshwar Bezawada[1], Kishore Kothapalli[2], Dugyala Raman[3], and Rui Li[4]

[1] Mahindra Ecole Centrale, Hyderabad, India. bru@mechyd.ac.in
[2] International Institute of Information Technology, Hyderabad, India. kkishore@iiit.ac.in
[3] Vardhaman College of Engineering, Hyderabad, India. raman.vsd@gmail.com
[4] College of Computer Science and Networking Security, Dongguan University of Technology of Science and Technology, Dongguan, China. ruli@dgut.edu.cn

Abstract. We focus on the problem of symmetric key distribution for securing shared resources among large groups of users in distributed applications like cloud storage, shared databases, and collaborative editing, among others. In such applications, resources such as data, are sensitive in nature and it is necessary that only authorized users are allowed access without the presence of on-line monitoring system. The *de-facto* approach is to encrypt a shared resource and deploy a key distribution mechanism, which enables only authorized users to generate the respective decryption key for the resource. The key distribution approach has two major challenges: first, the applications are dynamic *i.e.*, users might join and leave arbitrarily, and second, for a large number of users, it is required that the cryptographic technique be scalable and efficient. In this work, we describe an approach that overcomes these challenges by using two key techniques: first, flattening the access structure and applying efficient symmetric key distribution techniques. By flattening the access structure, we reduce the problem to that of key distribution of a resource among all the users sharing that resource. We consider this smaller flattened access structure and devise a unified key distribution technique that is sufficient for key distribution across all such structures. Our key distribution techniques have an important feature of a *public* secret and a *private* secret, which allows the group controller to publish updates to the keying material using the public secret and therefore, does not necessitate the users to be in constant communication with the group controller. Using this model we describe two efficient key distribution techniques that scale logarithmically with the group size and also handle group additions and removals. Furthermore, a user can be off-line for any amount of time and need not be aware of the dynamics of the system, which is important as it overcomes the problems posed by lossy channels. We have performed an experimental evaluation of our scheme against a popular existing scheme and show that they perform better for this scheme with the same security guarantees. As our approaches are easy to implement they are especially suitable for practical applications where security is viewed as an overhead rather than as a necessity.

1 Introduction

Motivation. Shared resource access is common in many application domains such as data access control in organizations, multi-level database applications like air-travel reservations, collaborative document editing and cloud file systems among others. The problem in such applications is to implement an efficient and scalable security mechanism which allows users selective access to the resources based on their privileges. This problem can be trivially solved if the environment is static i.e., the users of the application do not change or if the privileges of the users do not change over time. However, real-world applications are dynamic in nature and hence, the security mechanism must be able to handle such changes in an efficient manner without causing breaches of security. Thus, given the pervasive nature of such applications and the sensitivity of the data, there is a critical need to address the problem securing shared resources in real-time dynamic applications.

Problem Overview. A trivial solution to secure access to resources is to encrypt each resource with a unique symmetric key. For each resource to which the user has access, the user receives the set of decrypting keys for the resources. The trivial solution is simple and computationally efficient but has some drawbacks. First, the storage at the user is directly proportional to the number of resources he can access, for R resources the user stores R keys. Second, when a user is revoked from the system, the central authority needs to re-encrypt all the resources known to the revoked user and distribute the new keys to the remaining users who need them. In case of revocation, the cost of encryption is quite high for the central authority. Moreover, the central authority needs to communicate the changed keys individually to each of the remaining users in a secure manner. A more serious problem is that all users may not be online to receive the key updates. Thus, the trivial solution requires high user storage and has considerable communication overhead if users go offline for arbitrary periods of time.

Limitation of Prior Art. The existing approach to this problem is organize users and the resources in a hierarchical manner and define an ordering on the data access. For example, one such ordering is that users and data at the higher levels of the hierarchy can access data at lower levels of the hierarchy and so on. There have been several solutions [1–5, 7, 9, 17, 19, 21, 23, 28] that address the security of data in such access hierarchies. In these approaches, the data at level in the hierarchy is encrypted using a cryptographic key and users at higher levels in the hierarchy can *derive* the lower level keys in an efficient manner. These solutions reduce the user storage to only one cryptographic key $O(1)$ as the rest of the keys can be derived using this key. However, in many cases, the hierarchies can run deep and the key derivation time is considerable. Another major drawback in applying these solutions is that it is not trivial or possible to identify or generate an access hierarchy using the access control list. Thus, the main challenge in securing shared resource access is to identify the trade-off between storage and computational complexity.

Our Approach. From the above discussion, we note that, there is a need for a solution that can address a generic model of data sharing. We consider one such simple and generic model in which each resource is individually selected and all the users who can access this resource are grouped together. This model is natural in many applications. For applications that may have naturally defined access hierarchies, it is trivial to transform the hierarchy into such a *flattened* structure. The flat structure is well suited to model the type of sharing that occurs, say, in secure data bases and hence, our approach is applicable to a more general class of problems than access hierarchies.

Using the flat structure and the trivial solution described above, it is possible to reduce the computational complexity for deriving the necessary decrypting keys. However, this not only requires high storage at the users but also, the revocation and addition of users is non-trivial. Instead, we use a *public-space* based model where the decryption key of a resource is a function of a *public* secret and a *private secret*. The public-space model is simple in nature and consists of two pieces of information: public-information and private-information. The central authority encrypts the resource by using a combination of the public and the private information. At initialization, the central authority distributes the corresponding private-information to each user in a secure manner. Each user is given only the relevant private information and this information does not change regardless of the dynamics of the group. The decryption key of each resource is a function, F of the private-information held by all the users who have access to this resource. The public-information consists of one or more values evaluated over F on the private-information held by each user. Now, the central authority attaches the public-information to the resource as meta-data and stores it along with the resource. By using the necessary public-information and the stored private-information each user can locally derive the decryption key. If any change in the membership occurs, the central authority re-computes the public information using the private information of changed user group and updates the resource accordingly.

Technical Challenges and Solutions. The first technical challenge is to be able to generate the *flat* access structure from a given organization's access structure. We solve this by treating the access structure as a graph, where the nodes are associated with access levels and users, and then computing the transitive closure of this graph for each resource. The result is the set of users sharing a particular resource r, *i.e.*, the access control list of r and the set of resources accessible by a user, *i.e.*, the capability of a user.

The second technical challenge in our approach is to devise a key distribution scheme that does not require the user to store as many decryption keys as the number of authorized resources. Specifically, to design an efficient construction of the function F, that is dependant on the private-information of the users, and, on the derivation of the decryption key by the users. To reduce the storage at the users we use a logarithmic-keying approach where, for a set of R resources, each user needs to store only $O(\log R)$ keys.

Now, if a user is granted access to a resource, the central authority selects a unique subset of the keys from this pool of keys, derives the function F and the corresponding public-information. We design two key distribution approaches for implementing the function F. In the first approach, the central-authority randomly selects a secret polynomial which is of order $O(\log R)$. An arbitrary point on this polynomial is chosen as the encrypting key for the resource and made public. For each user, the central authority uses the keys of this user and evaluates them on the polynomial. All the evaluated values from all the users are published as public-information. To derive the decryption key, the user needs to interpolate the polynomial and evaluate it at the particular point published by the central authority. Since deriving a polynomial of order $O(\log R)$ requires $O(\log R)$ points, each user can easily compute the polynomial using the corresponding set of keys and the corresponding public-information.

In the second approach, the central authority computes the decryption key as an XOR of values derived using the keys selected from the users sharing the resource. For each user, the central authority uses the keys of this user and passes each key through a one-way function along with a public-value P that is specific to this user. The hashing is necessary to prevent leaking the actual values of the keys and the public-value acts as a salt for dictionary-based attacks. Now, the central authority XORs all the hash values from all the users. This combined hash value is used as the encryption key after appropriate expansion or reduction of the hash value length depending on the requirement. In order to derive this key, from the properties of XOR, each user needs the XOR of all the hash values contributed by the remaining users. Since, the basic key derivation operations are one-way hashing and XOR, this scheme is very efficient and can even be implemented in hardware.

Key Contributions. Our major contributions are as follows. (a) We devise an efficient key distribution approach for securing shared resource using a public-private information model. (b) We implement our model using two efficient key distribution approaches that only require the users to store a logarithmic number of keys to the number of authorized resources. (c) We show that the cost of handling user dynamics is better than existing approaches without compromising security. We have used key derivation cost, user storage, size of public-information and membership handling costs as metrics to evaluate our approaches. (d) The generic nature of our approach shows that more key distribution protocols are possible within this model and hence, there is further scope for expanding our approach to newer application domains.

Organization. The paper is organized as follows. In Section 2, we describe our system model and identify security requirements. In Section 3, we describe our framework in detail. We analyze the security of our framework in 5. In Section 4, we present the experimental results obtained from our framework and compare them with existing schemes. In Section 6, we conclude the paper and describe some future work.

2 System Model

In this section, we describe the problem background in detail and the system model. We also state our assumptions towards solving the problem. We conclude the section by describing some related work in this area.

2.1 Background

Applications with shared resources can be classified into two broad classes based on user behavior: those that require all the users to be online at the time of sharing, *e.g.*, video conferencing and those that do not have this requirement e.g., secure databases, file systems. For the sake of simplicity, we denote the former class of applications as *online* applications and the latter as *offline* applications. For *online* applications, the number of shared resources and the degree of sharing is small e.g., a group of users may subscribe to one or more multicast sessions. This problem has been studied extensively and many good solutions have been proposed in the literature [18, 25, 26]. In this work, we focus on the security of the *offline* applications and propose a framework to secure such applications.

2.2 System Model

We will describe our system model in terms of an access hierarchy and show how it can be transformed into a more general model. We assume that a central authority (CA) is in charge of access control, defines the access hierarchy and performs the key management tasks. The application has a set of m resources $R = \{R_1, R_2, \ldots, R_m\}$ in a resource store and a set of n users $U = \{u_1, u_2, \ldots, u_n\}$. The users are arranged in an access hierarchy where the access relationships are specified by the central authority. Formally, an access hierarchy of users is a partial order on the users so that $u_i \leq u_j$ if and only if u_j has access to every object that u_i has apart from the objects that u_j can access on its own. The users (and the resources) are grouped into security classes $SC_1, SC_2, \ldots, SC_n$ such that $SC_i \leq SC_j$ if $i \leq j$ *i.e.*, users belonging to class SC_j can access all resources that are accessible to users belonging to class SC_i and of other classes lower than SC_i. A user u_i is allowed to access a subset $S_i \subseteq R$ of resources. Moreover, if user u_i is above u_j in the hierarchy, then u_i is allowed to access all the resources in $S_i \cup S_j$. It is natural to view the hierarchy as a partial order $(U, \leq)$. We say that when user $u_i \leq u_j$, user u_j is allowed all the privileges associated with user u_i apart from what u_j has on his own. Such a hierarchy is best represented using its Hasse diagram which can be modeled as a directed graph $G = (V, E)$, where, V represents the users and E represent the access relationships. The resulting graph is acyclic in nature i.e., directed acyclic graph (DAG). We call this graph as the access hierarchy graph. We note that, in current literature, some special topologies of the hierarchy graph such as a tree [15, 20], graphs of a certain partial order dimension [2] are studied for simplicity. For any node $u \in G$ and $R \in S_u$, let $d(u, R)$ refer to the length of a shortest (directed) path from u to a node $v \in G$ because of which u can access

the object R. Let $d(u) = \max_{R \in S_u} d(u, R)$. The *depth* of a hierarchy graph G, denoted $d(G)$, is defined as $\max_{u \in V(G)} d(u)$.

Normally, in an access graph all the users at the higher levels can access all the lower level resources. However, there are some special cases of access hierarchies where the users at the higher layers are restricted from access all the lower level resources. This restriction is specified in terms of the depth of the hierarchy they can access and hence, such hierarchies are called limited-depth hierarchies. In the case of a mechanism that works for a limited depth hierarchy, we say that each vertex in the graph is associated with a number $\ell(v)$ that indicates that v is allowed to access resources that can be reached by a (directed) path of length at most $\ell(v)$. For a user u, we denote by $cap(u)$ the set of resources that u can access. Similarly, for a resource r, we denote by $acl(r)$ the set of users u such that $r \in cap(u)$. We extend this notation naturally when dealing with sets of users and resources.

2.3 Related Work

One approach to reduce the storage at the users is to use key derivation techniques [1–3, 5, 9, 17, 19, 21, 28] for access hierarchies. A key derivation technique can be briefly described as follows: a user belonging to a class SC_j is given some secret information and if any, public parameters. The resources belonging to a particular security class, SC_j are encrypted using a secret key SK_j. Now, if a user, belonging to SC_j, wishes to access the resources of some other class $SC_i \leq SC_j$, then, the user can use his secret information and any available public parameters to derive the decrypting key for the lower security class. In their seminal work, Akl and Taylor [1] proposed a scheme where keys are based on products of prime numbers. The scheme works on the underlying difficulty of finding factors for large prime numbers. As the prime numbers may get larger and hence, operations become more expensive, MacKinnon et. al. [17] relaxed the setting to not use prime numbers. However, the process of generating the required keys is still difficult. An improvement to these schemes is suggested by Chang and Buehrer [5] but relies on integer modulo exponentiation which is a costly operation. Other schemes using multiplicative properties of the modulo function are reported in e.g., [19]. Several efficient schemes which rely on key derivation mechanisms using symmetric cryptographic primitives, for example hash functions, are reported in the literature [3, 14, 16, 20, 27]. Schemes from [20, 27] do not adapt well to dynamism and require expensive updates to handle dynamism.

In [3], the authors describe an efficient key derivation scheme based on one-way hash functions where each class is given a key and it can derive a key of classes lower to it in the hierarchy using its own key along with some public information. However, we note that the cost of deriving a key can be directly proportional to the depth of the access graph. The dynamic access control problems such as addition/deletion of edges, addition/deletion of a class are also handled in an efficient manner by updating only the public information - a feature that was not available in earlier schemes. They also presented techniques to

minimize the key derivation time at the expense of user storage [2]. The idea is to add some extra edges and extra nodes called dummy nodes based on the dimension of a poset. However, as pointed out in [2], the computational complexity of finding the dimension of a given poset diagram is not known. Our approach provides a key derivation scheme which on an average performs comparable or better to [2, 3].

In [6, 11], the authors describe a scheme that combines techniques from discrete logarithms and polynomial interpolation. However, the user storage cost is high and support for dynamic operations requires costly polynomial interpolation. In [28], the authors present a scheme using polynomials over finite fields. However, the degree of the polynomial kept secret with the object store is very high. Key derivation also involves computations with such large degree polynomials and takes time proportional to the depth of access. Moreover, the cost of rekeying under dynamic updates is quite high. In [9], the authors attempt a unification of most of the existing schemes. This is done by identifying the central attributes of the schemes such as: node based, direct key based, and iterative.

Once database is treated as a service [13], it is easy to envision that database operations can be outsourced bringing in a host of security issues. In [10,24], the authors apply the key derivation approach of [3] to secure databases. However, to be able to use the key derivation schemes to problems like secure databases the access hierarchy needs to be built up-front. A virtual hierarchy of users is created and the scheme of [3] is used. But, as noted in [10,24], the computational cost of generating this virtual hierarchy can be quite high.

The key short-coming of the key derivation techniques is that applying these solutions is difficult if the access hierarchy is not available upfront or is not explicitly specified. Such a scenario occurs in secure data bases or in access control matrices, which implies that, in order to be able to use the key derivation techniques, the access hierarchy structure needs to be constructed for these applications. Although, techniques for constructing access hierarchies [10, 24] have been proposed, these are expensive and place additional pre-processing overhead on the system. The main advantage of the key derivation schemes is that the user storage is minimal $O(1)$. However, the computational complexity in key derivation scheme is proportional to the depth of hierarchy which can be $O(N)$ where N is the number of application users. Thus, the key derivation techniques reduce the storage complexity but increase the computation required to derive the required keys.

From this discussion, we note that, there is a possibility of trade-off between the user storage and the complexity of key derivation. Given these shortcomings, we note that, it is relatively easier to consider a hierarchy and *flatten* it before deploying any key distribution techniques. Flattening of a hierarchy simply means that we consider each resource individually and group all the users sharing that resource. Using this approach, we will be able to address those scenarios where the hierarchies are not readily available, which is the case in most practical user-level databases. Also, special hierarchies like, limited-depth hierarchies where resources given to a user fall between two levels of the hierarchy,

can also be addressed with a flattened hierarchy. Next, we describe our approach by which the access hierarchy graph is flattened thereby eliminating the need for the expensive pre-processing step. To secure the flat access structure, we describe storage efficient key distribution techniques that are based on the public-private model, which means that the decryption key of a resource is a function of some public information and the user's secret information.

3 Our Approach

In this section we describe the proposed framework for shared resource access. In Section 3.1, we describe our approach for flattening of the access hierarchy of a given organization. In Section 3.2, we describe our basic key distribution approach for securing resources known to a single user using the storage efficient logarithmic keying approach. In Section 3.3, we enhance our basic key distribution using Shamir's secret sharing technique and describe the solution to securing all resources shared by the users.

3.1 Flattening Access Hierarchies

The process of flattening the hierarchy can be seen as computing the transitive closure of the graph G. There are several algorithms for computing the transitive closure of a given directed graph [8]. Given the graph G, the transitive closure of G, denoted G^*, has at most n^2 edges if the graph G has n vertices. The central authority thus first computes the graph G^*. Note that, the process for computing transitive closure changes only slightly for limited depth hierarchies as the depth of each user is noted before computing the closure. The outcome of computing the transitive closure is that for every user u, $cap(u)$ is known and similarly for every resource r, $acl(r)$ is known.

3.2 Logarithmic Keying for Securing Single Owner Resources

Logarithmic keying refers to schemes that require users to store a logarithmic number of keys and achieve some security functionality. Such schemes have been in vogue [12,25] for reducing the cost of rekeying in secure group communication. In [12,25], the authors show that for N users $2 \log N$ keys at the group controller are sufficient to achieve the desired functionality. The key distribution is as follows: each user is assigned a $\log N$-bit identifier. Using this identifier, the group controller assigns to this user a unique $\log N$ sized subset from its pool of keys. In our scheme, we use a similar key distribution with some modifications to suit our requirements.

Now, to secure single owner resources with the logarithmic keying technique, each user stores atmost $O(\log m)$ symmetric keys where m is the number of resources to which he has access to. To encrypt a resource, the user selects a unique subset of keys from this subset, computes an XOR of the keys and uses this value to encrypt the resource. To enable sharing of resources, each

user who has access to the resource needs the decrypting key. We use Shamir's secret sharing approach to encode the encrypting (symmetric) key of the shared resource. The encrypting key can be locally computed using a subset of the $O(\log m)$ keys held by each user and using some public information. This public information can be stored in the resource store as resource meta-data.

We note that, the above key distribution approach can be generalized. Instead of choosing a fixed size subset of keys based on the binary identifier, the central authority can choose a unique but smaller subset of keys for each resource. Thus, the key distribution can be now stated as follows. The central authority generates a unique pool of keys for each user. From this pool of keys, for each $cap(u)$, the central authority selects a unique subset of keys to encrypt the resource. Since the subset of keys is unique by construction, no two resources will be encrypted with the same key. Moreover, it is clear the the pool of keys cannot be more than $O(2\log m)$ as it can be trivially shown that the number of subsets of size k i.e., $\binom{2\log m}{k}$, for some $k < \log m$, is greater than m, where k can be appropriately chosen using Stirling's approximation [8].

Reducing Storage Further. In the key derivation techniques [1–3, 5, 9, 17, 19, 21, 28] the user needs to store only one key, albeit, the cost of deriving the decryption key involves higher computation. We note that, the logarithmic keying can be replaced with a scheme which requires the user to store only one master key K. The scheme is as follows: to encrypt a resource R_i with identifier ID_i the central authority computes the encrypting keys as, $K_{R_i} = H_K(ID_i)$ where H denotes a secure one-way hash function. Since the user has the master key K, he can compute the encrypting key by performing one secure one-way hash computation. However, we note that, in the logarithmic keying scheme the user needs to perform $\log m$ XOR operations where m is the size of the resources. It can be seen that this computation is much faster than one secure hash computation even when m is as large as 2^{12}. Hence, this illustrates that reducing storage invariably increases the computation required for key derivation. From this discussion, we observe that, the logarithmic keying scheme reduces the key derivation overhead by increasing the storage complexity only slightly.

3.3 Securing All Shared Resources

The logarithmic keying approach works if a single user is only accessing the resources. To secure shared resources we apply Shamir's secret sharing scheme coupled with the logarithmic keying approach. We encrypt the resource using a secret that can be generated locally by the individual subsets of keys held by the users. Our final solution has two main steps.

Step 1: Key Distribution Notice that for each user u, it holds that $|cap(u)| \leq m$. To provide keys for the resources, the central authority, using the scheme described in Section **??**, picks $2\log m$ keys uniformly and independently at random from the field $\mathcal{F}$. Note that for all practical purposes, it can be assumed that the set of keys chosen for each user are all distinct. These $2\log m$ keys form the master keys for each user. For each user u and for every resource $r \in cap(u)$, the

central authority then allocates different subsets of size $\log m$ keys chosen independently and uniformly at random from the set of master keys at u. These are denoted as $S_u^r = \{K_{u,1}^r, K_{u,2}^r, \cdots, K_{u,\log m}^r\}$ and are called as the keys of resource r at user u. The central authority thus has $|acl(r)| \cdot \log m$ keys for resource r.

Step 2 : Key Management Given the key distribution from Step 1, we now apply ideas from Shamir's influential paper [22] to complete the solution. The central authority chooses a polynomial f_r of degree $\log m$ for resource r and uses this to encode the encrypting key for the resource. The encrypting key $k(r)$ of the resource r is computed as follows. The CA chooses a point on the polynomial f_r, say $p(r)$, and computes $f_r(p(r))$ and publishes $\langle p(r), f_r(p(r))\rangle$. The decryption key $k(r)$ is set to $f_r(0)$, and the polynomial $f_r(.)$ is kept secret by the CA. Now, the CA implements a $\log m$–out of–$|acl(r)| \cdot \log m$ threshold secret sharing scheme to enable a user to interpolate this polynomial. To this end, for each user $u \in acl(r)$, the CA evaluates f_r at each of the keys in S_u^r. These values are then made public. To access a resource r, a user u uses the subset S_u^r for r along with the public information, *i.e.*, evaluations of $f_r(.)$ on the key set S_u^r, for resource r to interpolate the polynomial $f_r(.)$. Finally, the user evaluates this polynomial at $f_r(0)$ to recover the encrypting key $k(r)$.

Storage and Computational Complexity. The storage complexity of each user is $O(\log m)$ and that of the central authority is $O(n. \log m)$ where n is the total number of users. We denote the average degree of resource sharing by m_r, *i.e.*, on an average m_r users share a particular resource. Given this information, the public information required per shared resource is given by $m_r \log m$. The complexity of interpolating a polynomial of degree $\log m$, for extracting the encrypted key, is $O(\log^2 m)$ operations using the well-known Lagrange's method. We note that, the main advantage of our scheme is the small degree of the polynomial compared to other schemes [6, 11, 28] based on polynomial interpolation.

Our framework allows for efficient updates to handle changes to the user set, changes to the hierarchy, and changes to the resource set. We now describe the operations required to address each of these events.

Addition of an user Suppose that the new user u along with a list of objects he can access is given. Let R_a denote the set of resources for which $acl(r)$ changes. The central authority chooses the set of master keys for u independently and uniformly at random from the field $\mathcal{F}$. For each resource $r \in cap(u)$, the CA also picks the subsets of keys S_u^r. Now, for every resource $r \in R_a$, the central authority evaluates the polynomial $f_r()$ at the points in S_v^r for every new member $v \in acl(r)$ and makes these evaluations public. No change to the polynomial or the key $k(r)$ of the object is required. Unlike other schemes, adding a user is very easy in our framework as only a few more evaluation of polynomials are required. We note that, adding a user in an access hierarchy can be easily modeled in our approach by considering the incremental transitive closure.

Revoking a User In this case, the central authority has to essentially change the polynomial for each of the affected resources. For every such resource, r, the CA chooses a different polynomial of degree $\log m$ and recomputes the public

information public for each user in $aclr$. The CA need not change $p(r)$, or the keys of the users but only has to change the encryption key of r.

Addition of an Authorization This corresponds to adding a resource r to $cap(u)$ for a user u. The CA associates a subset of keys S_u^r, evaluates f_r at the points (keys) in S_u^r, and the resulting values are made public.

Revoking an Authorization In this case, only one resource is affected. To handle this change, the central authority chooses a different polynomial and proceeds as in the previous case.

Addition of a Resource We now consider the case when a new resource r is added to the resource store. In this case, let us assume that $acl(r)$ is also provided. For each user u in $acl(r)$, the CA associates the set S_u^r and informs u of the same. The CA then chooses a polynomial $f_r(.)$ and computes the required public information.

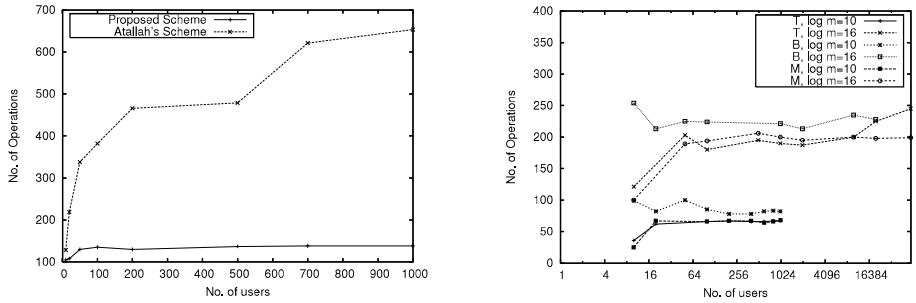
Extensions to Limited-Depth Hierarchies. Note that the above framework can work seamlessly for limited-depth hierarchies. Instead of finding the transitive closure of the access graph, we simply find the graph H such that $(u, v) \in E(H)$ if and only if $d_G(u, v) \leq d(u)$ and apply our approach.

4 Experimental Results

We performed three experiments. First, we compared the average number of operations required in our proposed framework against the scheme described in [3]. We refer to the scheme from [3] as *Atallah's scheme*. Second, to evaluate the efficiency of our framework in various settings, we describe a profiling of organizations. We evaluated our framework on each of these profiles. Finally, we compared the storage overhead of the proposed framework with that of *Atallah's* scheme. Implementation was in C++ on a general purpose Linux PC.

Comparison of Operations. We experimented with the number of users ranging from $n = 100$ and $n = 1000$. As we need to generate access graphs *Atallah's* scheme, we used random graphs with a diameter between $\log n$ and $2 \log n$. We used the transitive closure of the same graphs for evaluating our framework. The number of resources varies from 100 to 1000. To measure the average cost of accessing a shared resource, we computed the cost of accessing random resources at randomly chosen users using our framework and *Atallah's* scheme. For *Atallah's* scheme, we used SHA-1 as the chosen hash function as the derivation function. To move away from the specifics of the different implementation, the cost was measured by the average number of operations performed to derive the key of a randomly chosen resource. For our framework, the number of operations required to interpolate a set of points was measured. The results were averaged over 25 trials. In Figure 1(a), we show the results of the experimentation. We can see that on an average our framework requires a smaller number of operations as the size of the system grows.

Profiling and Efficiency of the Framework. We describe a practical profiling of different organizations that enable us to evaluate our framework in diverse settings. We note that our profiling can be used to evaluate other key derivation



(a) (b)

Fig. 1. Figure (a) shows comparison of computational cost of the our scheme with the scheme of Atallah et. al. [3]. Figure (b) shows the profiling results of our proposed scheme.

techniques as well. Our profiling is based on the distribution of users across the organizational hierarchy. The profiling is as follows: $Bottom - heavy$, $Top - heavy$, $Middle - heavy$, and $Uniform$. The $Bottom - heavy$ model corresponds to an organizational structure where there are a lot of users at the lower levels of the hierarchy. Similarly, a $Top - heavy$ model consists of more users at the top of the hierarchy; a $Middle - heavy$ model consists of more users in the middle of the hierarchy. In the $Uniform$ model, the users are equally spread across the organization. Note that, most organizations fall in these categories and hence, can be easily modeled by these profiles. We experimented on user sizes, $n = 2^{10}$ and $n = 2^{16}$. In Figure 1(b), we show the results of the experiments. For example, the line corresponding to, *T, log m=10*, means that the average number of operations were measure for a $Top - heavy$ organization for $n = 2^{10}$ users. In the figure, B stands for $Bottom - heavy$ and M stands for $Middle - heavy$. We can clearly see the variation in the number of operations. In the case of a $Top - heavy$ hierarchy where the degree of sharing is typically small, the number of operations even for 2^{16} users is around 200. This can be contrasted with a $Bottom - heavy$ hierarchy with a bigger degree of sharing. In this case, the number of operations increase but still is under 250. Predictably, the lines for the $Middle - heavy$ fall in between the Top and $Bottom - heavy$ cases.

Storage Comparison. In Figure 2, we show the public storage of our scheme against *Atallah's* scheme. For small hierarchies, our scheme requires a higher amount of public storage. But as the number of users increase, the public storage in our scheme is comparable to that *Atallah's* scheme. We note that, public-storage is necessary in such applications as it serves to reduce the amount of secure communication between the CA and the users.

5 Security Analysis of Our Framework

5.1 Soundness

Notice that as each resource is associated with an access polynomial which works along the lines of Shamir's secret sharing scheme, any valid user can always access

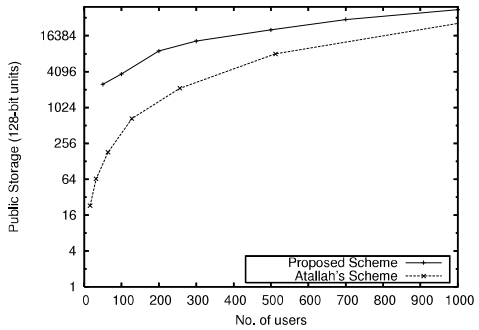


Fig. 2. The average public storage required in Atallah's scheme [3] and our scheme.

a resource. This holds because, the user needs to simply associate a subset of keys for the required resource from the keys in his key ring. Since the user knows the public information required to interpolate the resource polynomial, the user can access the resource.

5.2 Completeness and Collusion Resistance

Any scheme for shared resource access has to guarantee that a group of malicious users cannot pool up their secrets (keys) and derive access to any resource that they cannot otherwise access. If a solution is resistant to any group of up to k colluding users, then we call the solution to be k-collusion resistance. The parameter k is often called as the degree of collusion resistance of the solution. In the following, we show that our scheme is collusion resistant to a degree n where n is the number of users in the system.

To make the presentation formal, we need to define some notions. We follow the model introduced by Atallah et. al. [3]. We look at adversaries that can actively corrupt any node. When a node is corrupted by an adversary, it is possible for the adversary to get all the keys owned by the corrupted node. We also assume that keys assigned to the users are chosen uniformly at random from all possible keys in the field $\mathcal{F}$.

We let the adversary know the access graph G and its transitive closure. In effect, the adversary knows $\text{cap}(u)$ for every user u and $\text{acl}(r)$ for every resource r. For a given set C of corrupted nodes, let $\text{cap}(C) = \cup_{u \in C} \text{cap}(u)$. Let us fix any resource $r \notin \text{cap}(C)$ and the goal of the adversary is to access r. For this purpose, imagine an oracle $\mathcal{O}$ that knows the keys for $k(r)$ for r. The adversary creates a key (or a set of keys) $k(r^{'})$ and presents it to $\mathcal{O}$. The adversary is successful (wins) if $k(r) = k(r^{'})$.

While our description above uses an adaptive adversary, it can be noted that the power of an adaptive adversary is same as that of a static adversary. So in the rest of the presentation, we work with a static adversary. We call the above adversary as $\mathcal{A}$.

From the above description, it is clear that the advantage of the adversary $\mathcal{A}$ is tied to the ability to come up with the right polynomial. However, as stated in Shamir's paper [22], even if one point is not known it is difficult in the information theoretic sense to know the polynomial. In our case, the adversary $\mathcal{A}$ is not aware of any single point completely. It can know only the images but not the pre-images. For each of the possible $x \in \mathcal{F}$ for each of the pre-images, $\mathcal{A}$ can construct a polynomial. All these $|\mathcal{F}|^{\log m}$ polynomials are equally likely to be the correct polynomial for resource r. Hence, $\mathcal{A}$ cannot win with any non-negligible probability as $|\mathcal{F}|$ is large enough.

6 Conclusion and Future Work

In this paper, we presented a generic framework for securing shared resource access. We showed that our framework can be used for a general class of problems like access hierarchies and database security. Our framework used a logarithmic keying technique coupled with Shamir's secret sharing approach to reduce the computational complexity of encrypting and decrypting resources considerably. We also provided a profiling of organizations and evaluated our framework in these scenarios. The simplicity of our framework and our experimental results show that our framework can be easily deployed in practice.

We note that, however, our framework is meant for scenarios where there are many shared resources. Applications such as secure group communications have a limited number of shared resources with real-time requirements. Our framework can place considerable overhead in such applications and hence, would not be efficient. We are currently working on reducing the public storage in our framework and also, on the practical deployment of our framework in various applications.

References

1. Akl, S.G., Taylor, P.D.: Cryptographic solution to a problem of access control in a hierarchy. ACM Transactions of Computer Systems 1(3), 239248 (1983)
2. Atallah, M.J., Blanton, M., Frikken, K.B.: Key management for non-tree access hierarchies. In: Proc. of ACM SACMAT. pp. 11–18 (2006)
3. Atallah, M.J., Frikken, K.B., Blanton, M.: Dynamic and efficient key management for access hierarchies. In: Proc. of ACM CCS. pp. 190–202 (2005)
4. Castiglione, A., Santis, A.D., Masucci, B., Palmieri, F., Huang, X., Castiglione, A.: Supporting dynamic updates in storage clouds with the akltaylor scheme. Information Sciences 387, 56 – 74 (2017)
5. Chang, C.C., Buehrer, D.J.: Access control in a hierarchy using a one-way trap door function. Computers and Mathematics with Applications 26(5), 71–76 (1993)
6. Chen, T.S., Chen, H.J., How-Rernlina: A novel access control scheme based on discrete logarithms and polynomial interpolation. Journal of Ya-Deh University 8(1), 49–56 (1999)
7. Chu, C.K., Chow, S.S., Tzeng, W.G., Zhou, J., Deng, R.H.: Key-aggregate cryptosystem for scalable data sharing in cloud storage. IEEE transactions on parallel and distributed systems 25(2), 468–477 (2014)

8. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. McGraw Hill, 2 edn. (2001)
9. Crampton, J., Martin, K., Wild, P.: On key assignment for hierarchical access control. In: Proceedings of the 19th IEEE workshop on Computer Security Foundations. pp. 98–111 (2006)
10. Damiani, E., di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Selective data encryption in outsourced dynamic environments. Electron. Notes Theor. Comput. Sci. 168, 127–142 (2007)
11. Das, M., Saxena, A., Gulati, V., D.Pathak: Hierarchical key management schemes using polynomial interpolation. SIGOPS Operating System Review 39(1), 40–47 (2005)
12. Gouda, M.G., Kulkarni, S.S., Elmallah, E.S.: Logarithmic keying of communication networks. In: Proc. of the 8th International Conference on Stabilization, Safety, and Security of Distributed Systems. pp. 314–323. Springer-Verlag, Berlin, Heidelberg (2006)
13. Hacigümüs, H., Mehrotra, S., Iyer, B.R.: Providing database as a service. In: ICDE. pp. 29– (2002)
14. Jend, F.G., Wang, C.M.: A practical and dynamic key management for a user hierarchy. Journal of a Zhejiang University Science A 7(3), 296–301 (2006)
15. Liaw, H., Wang, S., Lei, C.: A dynamic cryptographic key assignment scheme in a tree structure. Computers and Mathematics with Applications 25(6), 109–114 (1993)
16. Lin, C.H., Lee, W., Ho, Y.K.: An efficient hierarchical key management scheme using symmetric encryptions. In: 19th International Conference on Advanced Information Networking and Applications (AINA'05). vol. 2, pp. 399–402 (2005)
17. MacKinnon, S.J., Taylor, P.D., Meijer, H., Akl, S.G.: An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. IEEE Transactions on Computers C-34(9), 797–802 (1985)
18. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In: CRYPTO. pp. 41–62 (2001)
19. Ray, I., Ray, I., Narasimhamurthi, N.: A cryptographic solution to implement access control in a hierarchy and more. In: Proc. ACM SACMAT. pp. 65–73 (2002)
20. Sandhu, R.S.: Cryptographic implementation of a tree hierarchy for access control. Inf. Process. Lett. 27(2), 95–98 (1988)
21. Santis, A.D., Ferrara, A.L., Masucci, B.: Cryptographic key assignment schemes for any access control policy. Infromation Processing Letters 92(4), 199–205 (2004)
22. Shamir, A.: How to share a secret. Communications of the ACM 22, 612–613 (1979)
23. Tang, S., Li, X., Huang, X., Xiang, Y., Xu, L.: Achieving simple, secure and efficient hierarchical access control in cloud computing. IEEE Trans. Computers 65(7), 2325–2331 (2016)
24. di Vimercati, S.D.C., Samarati, P.: Data privacy problems and solutions. In: in Proc. of the Third International Conference on Information Systems Security (ICISS). p. 180192 (2007)
25. Waldvogel, M., Caronni, G., Sun, D., Weiler, N., Plattner, B.: The versakey framework: Versatile group key management. IEEE JSAC (1999)
26. Wong, C.K., Gouda, M., Lam, S.S.: Secure group communications using key graphs. IEEE/ACM Transactions on Networking (2000)
27. Yang, C., Li, C.: Access control in a hierarchy using one-way functions. Elseveir Computers and Security 23, 659–664 (2004)
28. Zou, Z., Karandikar, Y., Bertino, E.: A dynamic key managment solution to acces hierarchy. International Journal Of Network Management 17, 437–450 (2007)