Design and Implementation of Privacy-Preserving, Flexible and Scalable Role-based Hierarchical Access Control

Tyler Phillips, Xiaoyuan Yu, Brandon Haakenson, Xukai Zou
Department of Computer Science
Purdue University School of Science
Indianapolis, Indiana 46202, USA
{phillity, xyu1, bhaakens, xzou}@iupui.edu

***Abstract*—In many domains, organizations must model personnel and corresponding data access privileges as fine-grained hierarchical access control models. One class of such models, Role-based Access Control (RBAC) models, has been widely accepted and deployed. However, RBAC models are often used without involving cryptographic keys nor considering confidentiality/privacy at the data level. How to design, implement and dynamically modify such a hierarchy, ensure user and data privacy and distribute and manage necessary cryptographic keys are issues of the utmost importance. One elegant solution for cryptography-based hierarchical access control combines the collusion-resistant and privacy-preserving Access Control Polynomial (ACP) and Atallah's Dynamic and Efficient Extended Key Management scheme. Such a model involves cryptographic keys used to encrypt data, can address confidentiality/privacy at the data level and can efficiently support dynamic changes to the RBAC access hierarchy. In this paper, we discuss several implementation challenges and propose solutions when deploying such a system including: data encryption and decryption, key storage and key distribution. Furthermore, we provide analysis of the efficiency and scalability of the resulting system.**

***Index Terms*—Cryptography-Based Hierarchical Access Control, Role-Based Access Control, Key Management, Secure Group Communication, Information Security**

I. Introduction

In recent years, the amount of data created, stored and leveraged by individuals and organizations has increased at a remarkable, exponential rate [20]. Within many domains, such as health-care and military domains, this data contains or reveals sensitive information. Therefore, this data must be secured and kept private from those not granted explicit, corresponding access privileges.

In order to provide a fine-grained, efficient mechanism to manage data access privileges, researchers have proposed and investigated many access control models over the years [10]–[16], [23]–[25]. One such class of access control models, Role-based Access Control (RBAC) models [11], [24], groups users and assigns privileges based on users' hierarchical roles within an organization. As many organizations naturally model personnel groupings and data privileges based on roles within the organization, RBAC models have seen wide acceptance and adoption. Unfortunately, the traditional RBAC model does not involve cryptographic keys, and does not consider privacy-preservation of users and data as a coherent part of its model. On the other hand, Cryptography-based (Hierarchical) Access Control (CHAC) models [3] have been proposed in order to directly address user and data privacy issues. However, how to equip RBAC with cryptographic keys, robustly address confidentiality/privacy issues and implement and deploy RBAC models in real-world, data-sensitive applications raises several additional important issues, such as key management and secure group communication. Furthermore, due to ever-increasing amounts of data, any viable solution must address each of these issues in such a way that supports robust efficiency and scalability.

In this paper, we propose a comprehensive access control system inspired by the Dual-Level Key Management (DLKM) scheme [33]. The proposed system augments an RBAC model with encryption and privacy-preservation capabilities through the combination of several techniques including: the Access Polynomial (ACP) technique [32] and Atallah's Dynamic and Efficient (Extended) Key Management scheme [4]. The resulting system provides a comprehensive solution that directly addresses the issues of access control, key management and secure group communication with fine-granularity. In addition to system design, we address several implementation details and analyze the efficiency and scalability of the resulting system.

In particular, this paper provides the following contributions:

1) A detailed presentation of how to employ Atallah's Scheme [4] in order to facilitate a Role-based Access Control (RBAC) model [11], [24].
2) A specific proposal of how to leverage the Access Control Polynomial (ACP) technique [32] in order to distribute a secure shared secret between group members efficiently and only as needed.
3) A detailed description of how to leverage the resulting system's key management scheme in order to perform encryption and decryption of sensitive data using two algorithms. The second algorithm, which we name Self-Authenticated Encryption/Decryption, is a novel method that allows the mapping between user groups (or roles) and corresponding data privileges to be kept private.
4) Identification of several other important implementation

978-1-7281-6741-1/19/$31.00 ©2019 IEEE
DOI 10.1109/TPS-ISA48467.2019.00015

challenges and the proposal of how to address them, including key storage and distribution strategies.

The paper is organized as follows: in Sec. II, we begin by outlining popular techniques which aim to address the issues of access control, key management and secure group communication. Next, in Sec. III, we outline the design of a comprehensive system which is able to robustly address each issue in an efficient and scalable manner. In Sec. IV, we provide multiple solutions, each with respective advantages and disadvantages, to several implementation challenges. Then, in Sec. V, we provide analysis of the computational and memory scalability of the system. Finally, in Sec. VI, we offer concluding remarks.

II. Related Work

Access control of sensitive data is an important and well-studied issue. Many access control models have been proposed and widely accepted in several domains. Discretionary Access Control (DAC) models [10], [25], where each user is given an explicit set of privileges, were once popular in commercial domains because of their flexibility and fine-granularity. Unfortunately, DAC models do not scale well as large numbers of users, each with their own set of privileges, become increasingly difficult to manage. Mandatory Access Control (MAC) models [15], [23] introduced levels of privileges for accessing the data objects in a system. In a MAC model, a user is assigned a privilege level and granted access to all the data objects of equal or lower privilege level. Unfortunately, MAC models are not well-fit for high security domains where many data objects should be accessible by only a small set of corresponding users. As the restrictions of DAC and MAC schemes were recognized, Role-based Access Control (RBAC) models [11], [24] saw wide acceptance and adoption. In RBAC models, privileges are assigned to groups of users. RBAC simplifies privilege management when a user's activities in the system change and also facilitates complex data privilege hierarchies with fine-granularity. As a result, RBAC models are well-suited for organizations which group their personnel in hierarchical roles. Other access control models, such as Relation-based Access Control (ReBAC) models [12], [13] and Attribute-based Access Control (ABAC) models [14], [16], have also been recently proposed. Due to the complexity of these recently proposed models, their acceptance and adoption has been limited. Currently, RBAC models are still the most widely accepted and deployed method of access control [29].

In order to provide data and user privacy, Cryptography-based (Hierarchical) Access Control (CHAC) models [3] have been proposed. In order to facilitate a CHAC model, additional issues, such as key distribution and management, must be addressed. Many proposed CHAC models and key management schemes can be used to facilitate an RBAC model. However, many of these schemes have scalability and design issues which prevent their application in demanding domains. Many schemes, including [3], involve division of two large primes which is computationally expensive as the number of bits in the primes increases. Other schemes often restrict the design of hierarchy to a tree-like structure [18], [26], [27]. This leads to these schemes being unfit for many domains which require flexible and complex hierarchies. Besides these early works, more comprehensive schemes [7], [8], [19], [31] have been proposed to support efficient modification operations on a hierarchy. Unfortunately, many of these schemes do not handle modifications locally within a hierarchy. This leads to a trusted group controller needing to re-compute and re-distribute keys to large sets of users upon hierarchy modifications. One elegant scheme, Atallah's Dynamic and Efficient (Extended) Key Management scheme [4], is able to facilitate arbitrary directed acylic graph (DAG) hierarchies. Furthermore, Atallah's Scheme handles modification operations locally within a hierarchy. This promotes system efficiency and scalability as there is much less need to re-compute and re-distribute keys when performing modifications.

Another important issue, secure group communication, addresses how to handle key synchronization among users in the same group within a hierarchical access control model. Many different protocols have been proposed, including: distributed group key distribution [2], distributed contributory group key agreement [6], decentralized group key management [21] and centralized group key distribution [17]. One interesting solution, the Access Control Polynomial (ACP) [32], is a provably privacy-preserving and attack-resistant method of distributing a shared secret to a group of users.

III. System Design

In this section, we discuss the design of a comprehensive system which robustly addresses the issues of access control, key management and secure group communication. The design of the proposed system is based upon the Dual-Level Key Management (DLKM) scheme proposed by [33]. The DLKM scheme addresses these issues at both the user-group and group-hierarchy levels. The user-group level of DLKM makes use of the privacy-preserving and collusion resistant Access Control Polynomial (ACP) [32] in order to distribute a shared secret among a group of users. The next level of the DLKM scheme involves the use of Atallah's Scheme [4] in order to build and efficiently facilitate a hierarchical Role-based Access Control (RBAC) model [11], [24].

The system involves cooperation and communication between a client, an authorization/key management server (AS) and a database/application sever (DS). The system assumes the AS is trusted as it will store all hierarchy data and contain all necessary keys to decrypt any encrypted data stored within the DS. Furthermore, we assume a trusted group controller (GC) exists who has access to the AS and is able to trigger modifications to the hierarchical RBAC model. We assume a secure communication channel exists between the client, AS and DS.

A. Access Control Polynomial

After grouping users based upon their role and data privileges within an organization, our system makes use of the ACP [32] in order to distribute a shared secret among a group's

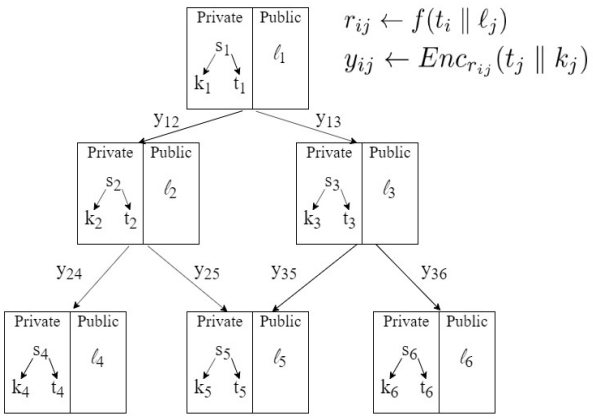

Fig. 1. Example hierarchical RBAC model built using Atallah's Scheme.

members. An ACP is a polynomial, $A(x)$, computed over finite field F_q where q is a large γ-bit prime:

$$A(x) = \prod_{i \in \Psi} (x - f(SID_i || z)) \tag{1}$$

In order to compute $A(x)$, each user ψ_i in group Ψ shares a private random integer input, $SID_i \in F_q$, with the AS. A user's SID could be derived by information stored on the AS during enrollment, e.g. the user's password hash. For each user ψ_i, the trusted AS then concatenates their SID_i with public value $z \in F_q$ and hashes the result by a public one-way function $f : \{0,1\}^* \rightarrow \{0,1\}^\gamma$. This results in $f(SID_i||z)$ for ψ_i.

The AS utilizes each user's $f(SID_i||z)$ to compute $A(x)$ as shown in Equation 1. The AS then uses the resulting $A(x)$ in order to compute public polynomial $P(x)$:

$$P(x) = A(x) + s \tag{2}$$

where $s \in F_q$ is a secret integer to be shared among users in group Ψ. As shown in Equation 2, s will be mixed with and masked by the constant term of $A(x)$ during the computation of $P(x)$. Finally, the AS publicizes $(z, P(x))$ to group Ψ by multicasting it to each user ψ_i. Each time the AS wishes to re-distribute $(z, P(x))$ to each $\psi_i \in \Psi$, it dynamically updates z as $z' \in F_q$, computes the resulting $A'(x)$ and $P'(x)$ and multicasts $(z', P'(x))$ to all ψ_i.

It can be seen that any user ψ_i in group Ψ can retrieve shared secret s by simply computing $s = P(f(SID_i||z))$. Furthermore, it can also be seen that public polynomial $P(x)$ is both collusion and privacy-preserving. Consider the case in which a proper subset of users, $\Phi \subset \Psi$, wish to collude to derive the private input, $f(SID_i||z)$, of user $\psi_i \notin \Phi$. Any member of the colluding subset, $\psi_j \in \Phi$, may compute the group's shared secret s by simply computing $s = P(f(SID_j||z))$. Using s, the colluding group can compute $A(x) = P(x) - s$, set $A(x) = 0$ and use a root-finding algorithm in order to compute all $f(SID_k||z)$ used to compute $A(x)$. Even in the most extreme case where the colluding group is made up of all group members besides the target group member, $\Phi = \Psi - \psi_i$, SID_i is computationally infeasible to derive from $f(SID_i||z)$ (assuming finite field F_q is sufficiently large).

These robust security and privacy features of the ACP can be further augmented by the inclusion of dummy values in the trusted AS's computation of $A(x)$:

$$A(x) = \prod_{i \in \Psi} (x - f(SID_i||z)) \prod_{j=1 \ldots d} (x - VID_j) \tag{3}$$

where the additional second term is made up of d dummy values $VID_j \in F_q$ randomly chosen by the AS. Without knowledge of the dummy values, it is impossible for any colluding subset Φ to successfully determine which roots of $A(x)$ are dummy values, VID_j, and which roots are user inputs, $f(SID_k||z)$.

B. Atallah's Scheme

After using the ACP to distribute shared secrets among group members, our system then organizes groups into a hierarchical RBAC model using Atallah's Scheme [4]. Atallah's Scheme involves modeling hierarchical relationships between user groups through the use of a directed acyclic graph (DAG), $G = (V, E, O)$, where V is a set of vertices of cardinatlity $|V| = n$, E is a set of edges of cardinality $|E| = m$ and O is a set of data objects of cardinality $|O| = p$. Atallah's Scheme requires the trusted GC to carry out hierarchy creation and maintenance. It also makes use of public hash function $f : \{0,1\}^* \rightarrow \{0,1\}^\gamma$ and symmetric encryption scheme ε. ε is made up of polynomial-time encryption function $Enc_{SK} : m \rightarrow c$ and decryption function $Dec_{SK} : \hat{c} \rightarrow \hat{m}$ where SK is an input encryption/decryption cryptographic key, m is a plaintext message and c is the ciphertext encryption of m.

Each vertex $v_i \in V$ represents a group of users, Ψ, in an organization with a corresponding role and data privileges. Each v_i is assigned a random public label, $l_i \in \{0,1\}^\gamma$, and uses a corresponding ACP in order to distribute a shared secret, s_i, to each of its group members, $\psi_j \in \Psi$. Using s_i, any of v_i's group members can derive two private cryptographic keys. The first private key, $k_i = f(s_i||0||l_i)$, where $k_i \in \{0,1\}^\gamma$ is used for data encryption/decryption. The second private key, $t_i = f(s_i||1||l_i)$, where $t_i \in \{0,1\}^\gamma$ is used for derivation of other groups' private keys.

Each object $o_i \in O$ represents a data object belonging to an organization which requires certain privileges in order

to access. Function $\mathcal{O} : V \rightarrow 2^O$ maps a vertex to a corresponding set of objects for which it is granted access such that $|\mathcal{O}(v_i)| > 0$ and $\forall i \forall j, \mathcal{O}(v_i) \cap \mathcal{O}(v_j) = \emptyset$ if and only if $i \neq j$. Each data object $o_j \in \mathcal{O}(v_i)$ is encrypted using k_i.

Each directed edge $(v_i, v_j) \in E$ is used to denote hierarchical relationships between user groups. Any directed edge, (v_i, v_j), from v_i to v_j requires two values. The first value, $r_{ij} = f(t_i \| l_j)$, is kept private and the second value, $y_{ij} = Enc_{r_{ij}}(t_j \| k_j)$, is publicized as (v_i, v_j)'s label. If a path exists between some vertex, v_i, to a descendant, v_j, v_i will be able to derive v_j's data encryption/decryption key, k_j, using only public group and edge labels as well as t_i (see Alg. 1). This will, in effect, grant any group v_i access privilege to any $\mathcal{O}(v_j)$ where v_j is a descendant of v_i.

For each node, v_i, Atallah's Scheme [4] defines the set of all descendants of v_i as $Desc(v_i, G)$ where G is the public components of the DAG. Likewise, Atallah's Scheme defines all ancestors of v_i as $Anc(v_i, G)$. We consider $v_i \in Desc(v_i, G)$ and $v_i \in Anc(v_i, G)$. This means v_i will be able to derive the private keys of any k_j where $v_j \in Desc(v_i, G)$ (but v_i will be unable to derive s_j). Furthermore, Atallah's Scheme defines the set of all immediate successors and predecessors of v_i as $Succ(v_i, G)$ and $Pred(v_i, G)$, respectively. We consider $v_i \notin Succ(v_i, G)$ and $v_i \notin Pred(v_i, G)$.

Finally, using these constructs, Atallah's Scheme [4] defines a $Derive(v_{source}, v_{target}, G)$ algorithm shown in Alg. 1. Using this algorithm, any user in group v_i is able to use her own t_i along with public node and edge labels in order to derive the private keys of any of its descendants $v_j \in Desc(v_i, G)$.

An example six node (or role) hierarchical RBAC model built using Atallah's Scheme can be seen in Fig. 1. For each node, v_i, only l_i and s_i must be stored by group members. The private keys, k_i and t_i, do not need to be stored by v_i and can instead be derived from s_i as needed.

C. Efficient Group and Hierarchy Modification Operations

As a result of the use of the ACP [32] and Atallah's Scheme [4], the resulting DLKM system supports efficient RBAC model modification operations. These operations include modifications at the group-hierarchy level as well as at the user-group level. Here, we discuss the details of how to perform each of these modification operations and their resulting computational complexities.

Insertion of a new node. When adding a new node, v_i, to the graph, v_i is first treated as if it does not have any edges connected to it or any users assigned to it. The GC creates the new node by computing and assigning its secret and public information. This involves assigning the new node v_i a random public label, $l_i \in \{0,1\}^\gamma$, and a random secret, $s_i \in \{0,1\}^\gamma$. Then, v_i's two private keys can be computed whenever they are needed as: $k_i = f(s_i \| 0 \| l_i)$ and $t_i = f(s_i \| 1 \| l_i)$. After assigning this secret and public information, connecting edges and new users can be sequentially added by use of the insertion of a new edge and user acceptance operations shown later in this section.

Algorithm 1: Atallah's Scheme method of deriving descendant keys.

```
Derive (v_source, v_target, G)
    if v_source = v_target then
        return v_source.get_k
    end
    if Path(G, v_source, v_target) = ∅ then
        return null
    else
        v_i = v_source
        k_i = v_source.get_k
        t_i = v_source.get_t
        for v_j ∈ Path(G, v_source, v_target) do
            r_ij = f(t_i || l_j)
            t_j || k_j = Dec_{r_ij}(y_ij)
            t_i = t_j
            k_i = k_j
        end
        return k_i
    end
```

Insertion of a new edge. Suppose v_i is to be assigned the privilege to access to $\mathcal{O}(v_j)$. A new edge, (v_i, v_j), must therefore be inserted into the graph so that v_i will be able to compute k_j. In this case, the GC must first compute $r_{ij} = f(t_i \| l_j)$, and then use the resulting r_{ij} to compute the public label of the new edge, $y_{ij} = Enc_{r_{ij}}(t_j \| k_j)$. These values are assigned to the new edge, (v_i, v_j), and, as a result, v_i is granted access to $\mathcal{O}(v_j)$. The GC should note that, in addition to v_i, all groups $v_a \in Anc(v_i, G)$ will also be granted access to $\mathcal{O}(v_j)$.

Deletion of an edge. The main security concern when performing deletion of an edge, (v_i, v_j), from the graph is that the group members of v_i may still be able to access $\mathcal{O}(v_j)$ if the deletion is not well performed. Suppose the edge (v_i, v_j) is going to be deleted from the graph by the GC. First, any public label, l_h, of v_j's descendants, $v_h \in Desc(v_j, G)$, should be reassigned a new random value, $l'_h \in \{0,1\}^\gamma$ in order to prevent access to $\mathcal{O}(v_h)$ by v_i or any of its ancestors, $v_k \in Anc(v_i, G)$. After updating public label l_h to l'_h for each $v_h \in Desc(v_j, G)$, v_h's two private keys will automatically updated as $k'_h = f(s_h \| 0 \| l'_h)$ and $t'_h = f(s_h \| 1 \| l'_h)$. This means secret information s_h does not need to be updated during an edge deletion. The updated private keys, k'_h and t'_h, will not be accessible by v_i or any of its ancestors as the path connecting them to $v_h \in Desc(v_j, G)$ will no longer exist after the deletion of (v_i, v_j) (assuming there exists no other path(s) connecting v_i or any of its ancestors to v_h). The detailed steps for carrying out an edge deletion are as follows:

1) For each node $v_h \in Desc(v_j, G)$, the GC must assign v_h a new random public label l'_h, and recompute k_h and t_h as mentioned above: $k'_h = f(s_h \| 0 \| l'_h)$ and $t'_h = f(s_h \| 1 \| l'_h)$.
2) Next, edges connected to any node $v_h \in Desc(v_j, G)$

should be updated by the GC according to the new labels $l_h\prime$ and private keys, k'_h and t'_h. For each v_h, the GC must find v_h's predecessors, $v_p \in Pred(v_h, G)$, and update any edge, (v_p, v_k), such that $r'_{ph} = f(t_p \| l'_h)$ and $y'_{ph} = Enc_{r'_{ph}}(t'_h \| k'_h)$. This will allow v_p and its ancestors to have the ability derive k'_h in the future and, as a result, have continued access to $\mathcal{O}(v_h)$.

Deletion of a node. There are three steps involved in deletion of a node v_i:

1) Using the edge deletion operation defined earlier, the GC sequentially deletes all edges (v_i, v_j) and (v_p, v_i) where $v_j \in Succ(v_i, G)$ and $v_p \in Pred(v_i, G)$. This will isolate node v_i such that when it is deleted, no ex-group member of v_i will be able to derive any key k_j.
2) Next, using the edge insertion operation discussed earlier, the GC must insert edges (v_p, v_j) for all $v_j \in Succ(v_i, G)$ and $v_p \in Pred(v_i, G)$. This will allow any of v_i's ancestors to have continued access to its descendants' private keys.
3) Finally, the GC must delete any record of v_i from the system, including its keys and public label.

Update secret key. If there arises a need to change a node v_i's secret information, s_i, four steps should be carried out:

1) First, the GC must assign v_i a new random secret, $s'_i \in \{0,1\}^\gamma$. As a result, the v_i's private keys will be updated based on the new secret key: $k'_i = f(s'_i \| 0 \| l_i)$ and $t'_i = f(s'_i \| 1 \| l_i)$.
2) Next, for each edge (v_p, v_i) where $v_p \in Pred(v_i, G)$, the GC must update $y'_{pi} = Enc_{r_{pi}}(t'_i \| k'_i)$.
3) Then, for each edge (v_i, v_h) where $v_h \in Succ(v_i, G)$, the GC first computes $r'_{ih} = f(t'_i \| l_h)$, and then updates $y'_{ih} = Enc_{r'_{ih}}(t_h \| k_h)$.
4) Finally, the GC must recompute its ACP based on the new secret s'_i. During this re-computation, public value z should be randomly assigned to a new value $z' \in F_q$. The resulting ACP re-computation is follows as:

$$A'(x) = \prod_{i \in \Psi}(x - f(SID_i \| z')) \prod_{j=1\ldots d}(x - VID_j)$$
$$P'(x) = A'(x) + s'$$

The GC then must send the new ACP to every user in group ψ to finish updating the secret key of the node.

User acceptance. Most of the operations above don't involve user-group level operations, which enables them to perform modifications efficiently. Suppose a new user with SID_n is going to join group v_i. No group-hierarchy level operations will need to be performed. Instead, the GC only recomputes v_i's ACP with new value $z' \in F_q$ as follows:

$$A'(x) = (x - f(SID_n, z'))$$
$$* \prod_{i \in \Psi}(x - f(SID_i \| z')) \prod_{j=1\ldots d}(x - VID_j)$$
$$P'(x) = A'(x) + s$$

User revocation. Suppose a user, ψ_i, leaves or is to be removed from their group, v_i:

1) To prevent the leaving user, ψ_i, from future access to the k_i or k_j where $v_j \in Desc(v_i, G)$, the user is first removed by the GC from v_i such that v_i's user group becomes $\Psi' = (\Psi - \psi_i)$.
2) Then, v_i's secret, s_i, should be updated using the update secret information operation described previously in this section. This will result in re-computation of v_i's ACP with the exclusion of removed user ψ_i's SID_i. As a result, ψ_i will be revoked future access to k_i.
3) Finally, the leaving user, ψ_i, must also be revoked access from any k_j where $v_j \in Desc(v_i, G)$. Each s_j does not need to be updated, but k_j does in case ψ_i stored them. Changing each k_j can be simply realized by changing v_j's public label, l_j, i.e., the GC updates l_j to l'_j and updating any public edge values, y_{jk}, connecting any two descendants of v_i. As any k_j and t_j are derived from both s_j and l_j, they will automatically be updated by these modifications.

TABLE I
TIME COMPLEXITY OF MODIFICATION OPERATIONS

Operation	Time Complexity
Insertion of a node	$O(1)$
Insertion of an edge	$O(1)$
Deletion of an edge	$O(n)$
Deletion of a node	$O(n)$
Key derivation	$O(n)$
Update secret key	$O(n + k^2)$
User acceptance	$O(k^2)$
User revocation	$O(n + k^2)$

In Table I, we provide the time complexity for each of the modification operations. Suppose there are $|V| = n$ nodes in total, with $|\Psi| = k$ users assigned to each node on average. Thus, traversal and derivation of $Desc(v_i, G)$ is an $O(n)$ operation as $Desc(v_i, G)$ may be made up of any subset of V. Multiplying each $(x - f(SID_i \| z))$ term, applying k modulo operations and additions during ACP computation results in a complexity of $O(k)$ operation. Thus, computing an ACP of size k is an $O(k^2)$ operation. Data communication between the server and all the users under in a group will take $O(k)$ at most. If multicast is possible, distribution of public ACP data will then be $O(1)$.

IV. IMPLEMENTATION CHALLENGES AND PROPOSED SOLUTIONS

The design of the two-level hierarchical access control scheme given in Sec. III is elegant, flexible, attack-resistant, and efficient. However, challenges exist when implementing the above design in real systems. These challenges, among others, include: (1) How to encrypt data fields such that users may only access the data for which they have the correct corresponding privileges, but also in such a way to facilitate decryption which can be performed correctly, practically and efficiently? (2) Besides SID_i, does a user need to store and carry other secret key(s), particularly, how can a user access their privileged data while moving and logging into the system from different computers at different locations? (3) How can the server distribute a shared secret to groups of users efficiently, regardless of whether multicast channels exist? We

will discuss each of these issues and propose practical and possible solutions below.

A. Data Encryption and Decryption

We define a plaintext dataset, $D \in \mathbb{R}^{r,c}$, as a matrix of r data records, each with c features. Likewise, we define an encrypted dataset as $C \in \mathbb{R}^{r,c}$. We consider each feature (or column) of the matrices as an object $o_j \in O$ which requires any user from group $v_i \in V$ to posses an access privilege in order to decrypt. An object (or column), o_{target}, of C can therefore be obtained and decrypted by v_i if and only if $o_{target} \in \mathcal{O}(v_{target})$ and $v_{target} \in Desc(v_i, G)$. We use notation D_{o_j} in order to obtain object (or column) o_j from matrix D. Furthermore, the value of object o_j for data record (or row) r can be obtained using the notation $D_{r;o_i}$. We assume all objects are assigned to a single corresponding group. If there exist objects that every group $v_i \in V$ should be able to access, these objects may be assigned to a single node, v_j, and then edges (v_i, v_j) can be inserted into the graph for each node $v_i \in V$.

We implement two schemes of data encryption and decryption. Both schemes employ MD5 [22] as their public hash function, $f : \{0,1\}^* \rightarrow \{0,1\}^\gamma$, and AES [9] using EAX mode [5] as their public symmetric encryption scheme, ε. One scheme assumes that the group-to-object mapping function, $\mathcal{O}$, is public. Therefore, given an object they wish decrypt, o_{target}, a member of group v_i can use public $\mathcal{O}$ in order to see any group with access to o_{target}: $v_h \in Anc(v_{target}, G)$ where $o_{target} \in \mathcal{O}(v_{target})$. Unfortunately, in some high security domains, such as health-care or military domains, it may be improper to publicize $\mathcal{O}$ and allow anyone to see which objects are accessible by which groups. Therefore, we design a second scheme which assumes $\mathcal{O}$ is hidden to everyone but the GC. In both schemes, the GC is responsible for encryption of the data, and users can later derive the desired data based only on public information and their secret keys.

1) Data encryption and decryption with mapping function $\mathcal{O}$ public: In this case, the GC performs encryption of D as follows:

- (Algo. 2, Line 3–5) For each node $v_i \in V$, the GC derives its private key, k_i, and finds all the data fields corresponding to objects $\mathcal{O}(v_i)$.
- (Algo. 2, Line 5–7) For each $o_j \in \mathcal{O}(v_i)$, the GC encrypts every row in the plaintext column D_{o_j} using k_i.

Decryption is straightforward when $\mathcal{O}$ is public. Suppose the user in group v_i wants to access a target object o_{target}:

- (Algo. 2, Line 3–6) Using the public mapping function, $\mathcal{O}$, the user should find the node v_{target} of which the o_{target} belongs to.
- (Algo. 2, Line 9–12) Next, the user determines if $v_{target} \in Desc(v_i)$. If not, access to data corresponding to o_{target} should be denied. Therefore, the user should not be able to derive the target node's key, k_{target}. Otherwise, the user can derive k_{target} using the derive key algorithm mentioned above (see Algo. 1).

Algorithm 2: Data encryption and decryption assuming group-to-object mapping function, $\mathcal{O}$, is public.

```
1  Encryption (G, D, O)
2      C ← D
3      for v_i ∈ V do
4          k_i ← v_i.get_k
5          for o_j ∈ O(v_i) do
6              for r ← 0 to D.rows do
7                  C_{r;o_j} ← Enc_{k_i}(D_{r;o_j})
8              end
9          end
10     end
11     return C
1  Decryption (G, C, o_target, O, v_source)
2      D_{o_target} ← C_{o_target}
3      v_target = null
4      for v_i ∈ Desc(v_source, G) do
5          if o_target ∈ O(v_i) then
6              v_target ← v_i
7          end
8      end
9      if v_target = null then
10         return null
11     else
12         k_target ← Derive(v_source, v_target, G)
13         for r ← 0 to C_{o_target}.rows do
14             D_{r;o_target} ← Dec_{k_target}(C_{r;o_target})
15         end
16         return D_{o_target}
17     end
```

- (Algo. 2, Line 13–14) Once the user derives k_{target}, they can decrypt the data row-by-row using k_{target}.

2) Data encryption and decryption without mapping information: When the users cannot access the group-to-object mapping function $\mathcal{O}$, there should be some additional information which users can employ to perform decryption correctly (if they have the correct privileges). We propose a novel encryption/decryption method dubbed Self-Authenticated Encryption/Decryption. The modified encryption process is as follows:

- (Algo. 3, Line 3–5) For each node $v_i \in V$, the GC derives its private key, k_i, and finds all the data fields corresponding to objects $\mathcal{O}(v_i)$.
- (Algo. 3, Line 5–7) For each $o_j \in \mathcal{O}(v_i)$, the GC encrypts every row in the plaintext column D_{o_j} using k_i. When the GC encrypts the r-th record in D_{o_j}, it encrypts the concatenation of the key and the actual data as: $Enc_{k_i}(k_i \| D_{r;o_j})$.

With the mapping function $\mathcal{O}$ private, no user is able to directly determine which node, v_{target}, corresponds to the object they wish to decrypt, o_{target}. Therefore, when trying to decrypt o_{target}, a user, $\psi_i \in v_i$, must sequentially derive and try all the keys k_j where $v_j \in Desc(v_i, G)$. The user will

Algorithm 3: Data encryption and decryption assuming group-to-object mapping function, $\mathcal{O}$, is private.

1 **Encryption** $(G, D, \mathcal{O})$
2 $\quad C \leftarrow D$
3 $\quad$ **for** $v_i \in V$ **do**
4 $\quad\quad k_i \leftarrow v_i.\text{get\_k}$
5 $\quad\quad$ **for** $o_j \in \mathcal{O}(v_i)$ **do**
6 $\quad\quad\quad$ **for** $r \leftarrow 0$ **to** $D.rows$ **do**
7 $\quad\quad\quad\quad C_{r;o_j} \leftarrow Enc_{k_i}(k_i \| D_{r;o_j})$
8 $\quad\quad\quad$ **end**
9 $\quad\quad$ **end**
10 $\quad$ **end**
11 $\quad$ **return** C
1 **Decryption** $(G, C, o_{target}, v_{source})$
2 $\quad D_{o_{target}} \leftarrow C_{o_{target}}$
3 $\quad v_{target} = null$
4 $\quad$ **for** $v_i \in Desc(v_{source}, G)$ **do**
5 $\quad\quad k_i \leftarrow Derive(v_{source}, v_i, G)$
6 $\quad\quad r \leftarrow 0$
7 $\quad\quad k_{target} \| D_{r;o_j} \leftarrow Dec_{k_i}(C_{r;o_{target}})$
8 $\quad\quad$ **if** $k_i = k_{target}$ **then**
9 $\quad\quad\quad v_{target} \leftarrow v_i$
10 $\quad\quad$ **end**
11 $\quad$ **end**
12 $\quad$ **if** $v_{target} = null$ **then**
13 $\quad\quad$ **return** $null$
14 $\quad$ **else**
15 $\quad\quad k_{target} \leftarrow Derive(v_{source}, v_{target}, G)$
16 $\quad\quad$ **for** $r \leftarrow 0$ **to** $C_{o_{target}}.rows$ **do**
17 $\quad\quad\quad k_{target} \| D_{r;o_{target}} \leftarrow Dec_{k_{target}}(C_{r;o_{target}})$
18 $\quad\quad$ **end**
19 $\quad\quad$ **return** $D_{o_{target}}$
20 $\quad$ **end**

know they have derived the correct k_{target} only when they they decrypt a field in $C_{o_{target}}$ and the decryption contains a matching k_{target} appended to plaintext data. Suppose the user in group v_i wants to access o_{target}. The user should carry out the following three steps:

- (Algo. 3, Line 4) The user first needs to get $Desc(v_i, G)$ from the public graph information.
- (Algo. 3, Line 4–13) For each node $v_j \in Desc(v_i, G)$, compute k_j and use k_j to decrypt the first record in within $C_{o_{target}}$. If the key contained in the decrypted data matches k_j, set v_{target} as v_j. If no key k_j of any $v_j \in Desc(v_i, G)$ matches the decrypted key, this means the user should be denied access to the data corresponding to o_{target}.
- (Algo. 3, Line 16–17) After k_{target} is found, the user can decrypt $C_{o_{target}}$ row-by-row with it. Within the decrypted data fields, the origin data can be seen appended to k_{target}.

In Table II, we provide time complexity of encryption and decryption for each scheme. Suppose the hierarchy DAG is balanced and there are $|V| = n$ nodes in total, each with $|\mathcal{O}(v_i)| = m$ data privileges on average. In both methods, encryption will take $O(nm)$ time as the GC has access to all keys stored in the AS and the node-to-object mapping, $\mathcal{O}$. For decryption, both schemes first require that a user in v_i spend $O(n)$ time finding the the set of nodes $Desc(v_i, G)$. In the case of the first scheme, the user in v_i can then check if the target object they wish to decrypt in access belongs to any node $v_j \in Desc(v_i, G)$. If so, she must spend an additional $O(\log(n))$ time to derive the corresponding target key along the path of descendants. In the case of Self-Authenticated Encryption/Decryption, $\mathcal{O}$ is not public so the user in v_i must spend $O(n)$ time to try all keys they can derive from $Desc(v_i, G)$. As a result, both schemes have an overall complexity of $O(n)$ to gather the necessary keys before decryption, but the Self-Authenticated Encryption/Decryption will take $O(n)$ to perform decryption without public $\mathcal{O}(v_i)$, while the first scheme will take $O(\log(n))$ with public $\mathcal{O}(v_i)$.

TABLE II
TIME COMPLEXITY OF DATA ENCRYPTION AND DECRYPTION

	Public Mapping		Private Mapping	
	Encryption	Decryption	Encryption	Decryption
Time Complexity	$O(nm)$	$O(n + \log(n))$	$O(nm)$	$O(2n)$

B. Key Storage

Key storage is another important implementation issue. As mentioned in Sec. III, the AS will house all keys and hierarchy data. The trusted GC will have access to the AS and will perform all key management operations. Meanwhile, the DS will contain all the encrypted data for which user can be granted access. The AS will serve as an intermediary between clients and the DS.

Through our use of the ACP technique [32], we assume each user's SID is stored by the server for quick re-computation and re-distribution of $A(x)$ and $P(x)$. Unfortunately, this makes the AS a single point of failure. While it can be made computationally infeasible for an attacker to derive $f(SID_i \| z)$ from the $P(x)$, the attacker could instead try to infiltrate the AS. In the case the attacker is successfully able to infiltrate the AS, they will be able to retrieve all encryption/decryption keys along with all the SID values. While system design assumes all encryption/decryption keys must be stored on the AS, it is possible for the SID values to be stored elsewhere. Here we describe two key storage strategies: (1) storing each user's SID in the AS and (2) storing each user's SID on the client. For both strategies, it should be noted that, as mentioned in Sec. III, we assume a secure communication channel exists between the client AS and DS.

1) Storing Keys on Server: In Sec. III, we assume that each user's SID is derived from information, such as a hash of the user's password, and stored on the AS during enrollment. In this way, both the client and the server will have access to the SID of a user. This means that when a user ψ_j is to be added to a group, the server will simply add ψ_j's SID_j to the ACP computation for this group (as all other necessary SIDs are already stored and possessed by the AS). The AS updates

public value z, computes $A(x)$ and adds s_i in order to get $P(x)$.

At this point the AS can send $(P(x), z)$, and the public components of the RBAC DAG graph, G, to the client. Since the client knows their own login information, they can derive their SIDs and hash it with public value z. Plugging the resulting value into $P(x)$ will result in the secret key s_i if and only if the user is a valid member of the group. Then, the client can store their secret s_i, use it to derive it's private key k_i and t_i, and use it to derive other private keys.

When a user would like to access data, they can send a request for the data they desire along with necessary private keys to the AS. The request and parameters can then be forwarded to the DS. The DS will use this information to decrypt the requested objects and send the resulting decrypted information back to the client. An example of this protocol can be seen in Fig. 2.

If the DS is not trustworthy, the AS could simply request the encrypted data object from the DS. Then, the trusted AS could use the client's provided key to decrypt the data and forward it to the client. The AS could also instead forward the encrypted data to the client and allow the client to decrypt the data herself (in which case the client would only need to indicate the target data object, o_{target}, and not provide the AS with a decryption key, k_{target}).

When storing the secret s_i, the client must know if they have the most up-to-date version. In some situations, such as user revocation, the secret s_i must be updated by the GC and AS. The client must be made aware of this change. One solution would be for the AS to assign secret s_i a version number and increment the version number after updates. Then, when sending the ACP information, the AS could also send the client the corresponding version number. The client may then ask the server for the current version number upon login. If the version the client is storing differs from the server's version, the client then knows that they must request the updated ACP information in order to calculate the new secret s_i.

2) Storing Keys on Client: In this section, we discuss how to ensure the privacy of the SIDs by storing the SIDs only on the client side. In this scheme, after any user ψ_i's enrollment, the user needs to pick a random number $z_i \in (0,1)^\gamma$, compute $f(SID_i||z_i)$ and store SID_i and z_i. When the user is added to some group, the AS will need the user to send their $f(SID_i||z_i)$, and then AS can then add $f(SID_i||z_i)$ to its ACP computation. This way, even if the AS is compromised, an attacker will only obtain the set of $f(SID_i||z_i)$ used in the ACP computation. As noted previously, it is computationally expensive to retrieve the original SID_i values from the $f(SID_i||z_i)$ hashes. To further augment system security, the AS can discard the $f(SID_i||z_i)$ values after ACP computation. In this case, the AS will need to request all the users in the group resend $f(SID_i||z_i)$ when re-computation of ACP is necessary. If some users are offline or unable to send back their hashed SIDs in a certain amount of time, the server will just compute a new ACP based on the SIDs it receives. The absent user(s) can then re-join the group by sending their $f(SID_i||z_i)$ value(s) the next time the server requests them. This may reduce the size of ACP as some group members are sometimes excluded, but as long as there is at least one user involved in the computation, their SIDs and s_i will be hidden by the ACP dummy values, VID_j.

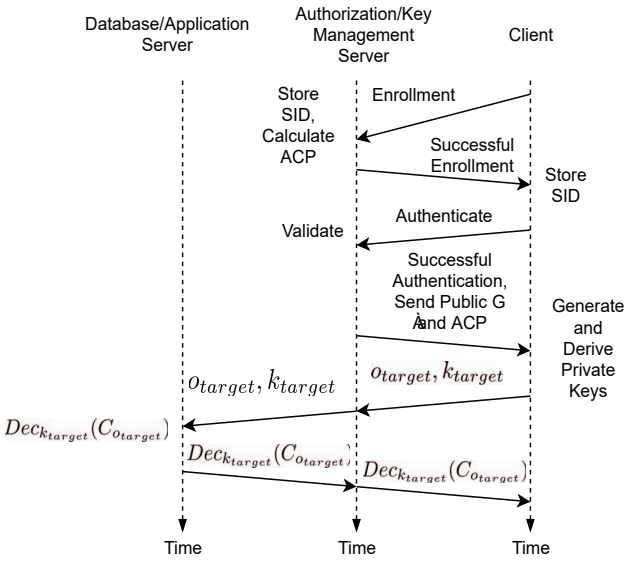

Fig. 2. Client, key management/authorization and application/database server carrying out enrollment, authentication and decryption protocols.

The AS only needs to publicize $P(x)$, the public components of G, and the version number of the secret s_i to the client. If a user has a different key version number, she must request that the AS recompute $P(x)$ with the inclusion of her hashed SID. Otherwise, she must plug their pre-computed $f(SID_i||z)$ into $P(x)$ to s_i. In order for this scheme to work, it must be assumed that those users requesting ACP should be able to send their hashed SIDs to the server. This will require some coordination from clients and the AS.

In either of the two schemes, the system can support remote access from different machines and locations. Assuming the user is able to provide their SIDs to the client, authorization and key derivation should work as previously explained.

Users accessing the system from devices with little computational power can also be accommodated by the AS. As long as the AS server stores either the plain SIDs or hashed SIDs, the server could carry out key derivation and decryption on behalf of the client. If this were to occur, robust authentication of users would be necessary before performing decryption for the user or providing the user the resulting decrypted data.

C. Key Distribution

One important concern is how information is distributed between the client and the AS. Ideally, it would be best to use multicast to send public information to the clients. Thus, public information for each group of users would only have to be sent once. If unicast is used, then the AS will have to establish a connection between every user and send information to each one individually. This does not scale well, particularly in organizations with roles containing a large number of users.

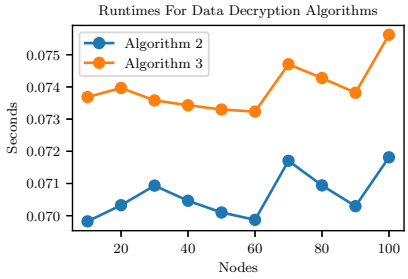

Fig. 3. Computational analysis of encryption/decryption Algorithm 2 and Algorithm 3 as graph size increases.

However, multicast has some drawbacks that could lead to unicast being a better choice despite it's inefficiency. Generally, multicast is only implemented on local area networks (LANs) and may require additional network configuration before it can be used. This is fine if the data within the DS is intended to only be accessible at one physical location, but this will create problems if the data within DS is intended to be accessed over the internet.

For most applications it is likely that clients will need to be able to communicate with the AS and DS over the internet. In this situation, unicast is likely the better choice as it requires no additional setup and can be used to communicate with clients over the internet. While unicast is less efficient, it is easier to use and will provide the client with more mobility. If unicast is used, any unicast packets must be encrypted so that information is not sent in plaintext. This can be solved by the use of TLS/SSL which will create an encrypted link between client, AS and DS. Securing the transport layer in this way is a standard and widely adopted practice that will protect the data from being deciphered if it is intercepted.

V. Experiment

In this section, we offer computational and memory scalability analysis of the resulting system. Each experiment was ran on a simple Lenovo Thinkpad 13 Ultrabook laptop with a Intel i5-6200U CPU and 8GB of RAM. Both experiments suggest that the experiment is quite scalable and can easily accommodate organizations which organize themselves into complex hierarchies with hundreds of roles and thousands of personnel members.

A. Data Decryption Experiment

In Sec. III, we offer theoretical analysis of the computational complexity of many different hierarchy modification operations. Decryption of target data objects needs to be performed in real-time when deployed in many real-world systems. We therefore examine the run-times of the two proposed decryption methods (see Algo. 2 and Algo. 3 in Sec. IV) as hierarchy DAG size increases. We performed this experiment by beginning with a 10-node graph and incremented the number of nodes until we reached a 100-node graph. Each node added to the graph was assigned a group of 100 users. For each graph size, we randomly generated a corresponding DAG with random edges. It should be noted that we did not add any random edges which would create a cycle in the graph. Then, for each randomly created DAG, we selected a random source node and target object belonging to one of the source node's descendants. The source node carried out decryption using both methods. For each graph size, we repeated each step three times and recorded the average of the results.

The results of the experiment can be seen in Fig. 3. The x-axis represents the number of nodes used while the y-axis represents the amount of time in seconds to complete the decryption algorithm. The first version of the decryption algorithm uses the object mapping technique of Algo. 2 while the second version of the decryption algorithm uses the Self-Authenticated Decryption method of Algo. 3.

Due to the random nature of our experiment, decryption time does not monotonically increase as DAG size increases. For all of the DAG sizes, Algo. 3 has higher run-times. This is to be expected as Algo. 3 assumes the object mapping is private. Therefore, the keys of all descendants of the source node must be computed and used during decryption. As a result, it is certain that Algo. 3 will take as long (in the best case) or longer than Algo. 2. Fortunately, Algo. 3 only takes an additional ~0.005sec longer than Algo. 2 on average. Depending on the domain the proposed system is deployed in, this additional ~0.005sec may be a worthwhile trade-off in order to keep the group-to-object mapping, $\mathcal{O}$, private. It should be noted that the ~0.005sec additional time for Algo. 3 is not fixed. There may be cases where Algo. 3 takes far greater time than Algo. 2. This would be dependant on the path length from some source to target node and the amount of descendants of the source node. It should also be noted that Algo. 3 has the disadvantage that a user must brute-force try all descendant keys before realizing they do not have the correct permissions to access a target data object (as the group-to-object mapping, $\mathcal{O}$, private). This differs from Algo. 2 in that a user can utilize their set of descendants and the public group-to-object mapping, $\mathcal{O}$, in order to directly determine if they have access to a target data object.

Both algorithms are able to consistently decrypt data within a tenth second on the simple Lenovo laptop, even when the graph contains hundreds of nodes and thousands of users. This provides strong indication that the system can scale easily and be used by large organizations.

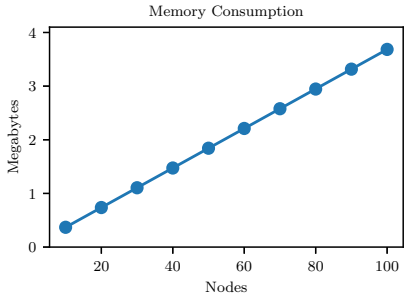

Fig. 4. Memory usage required as DAG size increases.

B. Memory Consumption Experiment

The experiment provided analysis of memory consumption by the system. Again, the x-axis represents the number of nodes in the randomly generated DAG. The y-axis represents

the amount of megabytes used for this DAG object and all associated data in memory.

The results of this experiment can be seen in Fig. 4. The memory consumption of the DAG clearly grows linearly with respect to size of the graph. This is because each node being added to the graph requires storage of a fixed set of data. Each node being added requires storage of: a secret key s_i, a public label l_i, edges, users, and an ACP. In this experiment the amount of users per node is kept constant so the amount of bytes will not vary much but it is to be expected that node memory consumption will depend on the amount of users. Since the amount of users in each node is the same, memory consumption increases linearly. At the largest DAG size of 100 nodes and 10,000 users, the required memory consumption is less than 4MB. This small memory consumption also provides strong indication that the system can scale well to large organizations.

VI. Conclusion

In this paper, we presented the system design, implementation details and scalability analysis of a fine-grained, efficient and privacy-preserving hierarchical key management system. The system supports fine-grained access control and efficient modification operations at both the user-group level through the use of the ACP technique [32] and at the group-hierarchy level through the use of Atallah's Scheme [4]. This fine-granularity and efficiency, paired with the robust computational and memory scalability demonstrated by our experiment, illustrates that the proposed system is suitable for deployment in demanding, complex real-world applications. In addition to system design, we have also discussed several implementation challenges and solutions such as: data encryption and decryption, key storage and key distribution. For each of these issues, we have offered multiple solutions, each with corresponding advantages and disadvantages. A system designer may use these insights to fine-tune the system to the needs of their specific application. Our Python code implementation is provided for open use at [1].

Acknowledgment

This work is partially supported by a U.S. National Science Foundation grant (CICI #1839746) and also the NSF Jetstream [28]/XSEDE [30] project (ACI-1445604 & OCI-1053575). We thank Mr. Jeremy Fischer and Mr. David Hancock for their assistance with allocating needed computing resources and porting the developed VM image on JetStream, which was made possible through the XSEDE Extended Collaborative Support Service (ECSS) program.

References

[1] *https://github.com/phillity/atallah_acp/*.
[2] P. Adusumilli, X. Zou, and B. Ramamurthy. Dgkd: Distributed group key distribution with authentication capability. In *Proc. of 6th Annu. IEEE SMC Infor. Assu. Workshop*, pages 286–293, 2005.
[3] S. Akl and P. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM TOCS*, 1(3):239–248, 1983.
[4] M. Atallah and M. Blanton et al. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):18:1–43, 2009.
[5] Mihir Bellare, P. Rogaway, and D. Wagner. Eax: A conventional authenticated-encryption mode. *IACR Eprint archive*, 2003.
[6] H. Chan and V. Gligor et al. On the distribution and revocation of cryptographic keys in sensor networks. *IEEE Transactions on dependable and secure computing*, (3):233–247, 2005.
[7] Tzer-Shyong Chen and Jen-Yan Huang. A novel key management scheme for dynamic access control in a user hierarchy. *Applied Mathematics and Computation*, 162(1):339–351, 2005.
[8] Hung-Yu Chien and Jinn-Ke Jan. New hierarchical assignment without public key cryptography. *Computers & Security*, 22(6):523–526, 2003.
[9] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.
[10] D. Downs and J. Rub et al. Issues in discretionary access control. In *IEEE Symposium on Security and Privacy*, pages 208–208, 1985.
[11] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.
[12] Philip WL Fong. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 191–202. ACM, 2011.
[13] Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. Relbac: Relation based access control. In *2008 Fourth International Conference on Semantics, Knowledge and Grid*, pages 3–11. IEEE, 2008.
[14] Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015.
[15] Y. Jiang and C. Lin et al. Security analysis of mandatory access control model. In *IEEE Inter. Conf. on SMC*, volume 6, pages 5013–5018, 2004.
[16] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *IFIP Annual Conf. on Data and Applications Security and Privacy*, pages 41–55, 2012.
[17] Xiaozhou Steve Li, Yang Richard Yang, Mohamed G Gouda, and Simon S Lam. Batch rekeying for secure group communications. *group*, 1:9, 2001.
[18] HT Liaw, SJ Wang, and CL Lei. A dynamic cryptographic key assignment scheme in a tree structure. *Computers & Mathematics with Applications*, 25(6):109–114, 1993.
[19] Chu-Hsing Lin. Hierarchical key assignment without public-key cryptography. *Computers & Security*, 20(7):612–619, 2001.
[20] Bernard Marr. How much data do we create every day? the mind-blowing stats everyone should read. *https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read*, May 21, 2018. Accessed in August 2019.
[21] Suvo Mittra. Iolus: A framework for scalable secure multicasting. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 277–288. ACM, 1997.
[22] R. Rivest. The md5 message-digest algorithm, 1992.
[23] Patrick Sack, Edward Austin, and Scott Gaetjen. Mandatory access control label security, November 9 2010. US Patent 7,831,570.
[24] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.
[25] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.
[26] Ravinderpal S Sandhu. On some cryptographic solutions for access control in a tree hierarchy. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, pages 405–410. IEEE Computer Society Press, 1987.
[27] Ravinderpal S Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27(2):95–98, 1988.
[28] C. Stewart, T. Cockerill, I. Foster, and D. Hancock et al. Jetstream: a self-provisioned, scalable sci. and eng. cloud environment. *XSEDE'15 Conf.: Sci. Adv. Enabled by Enhanced Cyberinfra.*, pages 1–8, 2015.
[29] Vivy Suhendra. A survey on access control deployment. In *FGIT-SecTech*, 2011.
[30] J. Towns and T. Cockerill et al. XSEDE: Accelerating scientific discovery. *Computing in Science & Engineering*, 16(5):62–74.
[31] Sheng Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8):750–759, 2002.
[32] X. Zou, Y. Dai, and E. Bertino. A practical and flexible key management mechanism for trusted collaborative computing. In *IEEE INFOCOM*, pages 538–546, 2008.
[33] Xukai Zou, Yuan-Shun Dai, and Xiang Ran. Dual-level key management for secure grid communication in dynamic and hierarchical groups. *Future Gener. Comput. Syst.*, 23(6):776–786, July 2007.