User-Friendly Design of Cryptographically-Enforced Hierarchical Role-based Access Control Models

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

***Abstract*—Data access control is a critical issue for any organization generating, recording or leveraging sensitive information. The popular Role-based Access Control (RBAC) model is well-suited for large organizations with various groups of personnel, each needing their own set of data access privileges. Unfortunately, the traditional RBAC model does not involve the use of cryptographic keys needed to enforce access control policies and protect data privacy. Cryptography-based Hierarchical Access Control (CHAC) models, on the other hand, have been proposed to facilitate RBAC models and directly enforce data privacy and access controls through the use of key management schemes. Though CHAC models and efficient key management schemes can support large and dynamic organizations, they are difficult to design and maintain without intimate knowledge of symmetric encryption, key management and hierarchical access control models. Therefore, in this paper we propose an efficient algorithm which automatically generates a fine-grained CHAC model based on the input of a highly user-friendly representation of access control policies. The generated CHAC model, the dual-level key management (DLKM) scheme, leverages the collusion-resistant Access Control Polynomial (ACP) and Atallah's Efficient Key Management scheme in order to provide privacy at both the data and user levels. As a result, the proposed model generation algorithm serves to democratize the use of CHAC. We analyze each component of our proposed system and evaluate the resulting performance of the user-friendly CHAC model generation algorithm, as well as the DLKM model itself, along several dimensions.**

***Index Terms*—Cryptography-based Hierarchical Access Control, Role-based Access Control, Key Management, Usability**

I. Introduction

Much of the data generated, recorded and leveraged by organizations encodes confidential, sensitive information. Unfortunately, many such organizations are often the victims of large-scale data breaches in which attackers steal or expose private data. Such exposure of sensitive data often yields substantial societal, financial and legal ramifications [17]. Therefore, organizations must ensure that sensitive data is only accessible by the set of users and personnel given appropriate access privileges. In fact, in high-security domains, such as healthcare [18] and government [23], organizations may be explicitly required by laws to protect the privacy and confidentiality of sensitive data.

Fortunately, much research has been performed in order to deliver methods for organizations to exercise fine-grained data access controls [26]. Many access control models have been proposed in order to define and enforce access control policies in a systematic manner [10]–[16], [25], [27], [28]. One such access control model, the Role-based Access Control (RBAC) model [27], assigns access privileges to groups of users based on their role within an organization. Due to the RBAC model's flexibility and simplicity, it has been widely accepted and deployed [32].

Unfortunately, the traditional RBAC model does not include the use of cryptographic keys in order to enforce access control policies and ensure the confidentiality of sensitive data [27]. Therefore, Cryptographic Hierarchical Access Control (CHAC) models [3] have been proposed. CHAC models introduce the use of key management schemes in order to facilitate storage, distribution and usage of cryptographic keys. The resulting cryptographic keys, in turn, can then be used to facilitate a hierarchical RBAC model and explicitly enforce data privacy [3], [20], [29], [30]. One recently proposed and comprehensive key management scheme, the Dual-Level Key Management (DLKM) scheme [24], [36], combines the Access Control Polynomial (ACP) [35] and Atallah's Efficient Key Management scheme [5] in order to provide cryptographically-enforced fine-grained access controls and privacy at both the data and user levels. The collusion-resistant ACP technique is used to distribute shared secrets to groups of users sharing a role within an organization. Atallah's scheme is then used to organize such groups of users into a cryptographically-enforced, hierarchical RBAC model in which dynamic modifications can be handled efficiently.

On major issue facing CHAC models is that their key management schemes, such as the DLKM scheme, are often represented by tree or directed acyclic graph (DAG) structures which cannot be easily interpreted or understood by users without intimate knowledge of symmetric encryption, key management and hierarchical access control models [3], [5]. Therefore, we propose a model generation algorithm which serves to effectively democratize cryptographically-enforced hierarchical RBAC models. More specifically, in this paper we provide the following contributions:

- We provide the specific details of how each component of the DLKM scheme is leveraged in order to facilitate a privacy-preserving, flexible and efficient hierarchical RBAC model.
- We design a novel CHAC generation algorithm. The

978-1-7281-6607-0/20/$31.00 ©2020 IEEE

algorithm uses a highly user-friendly representation of access control policies as input and yields a corresponding DLKM-based model.
- We implement both the CHAC generation algorithm and the DLKM scheme in order to evaluate their performance. Our experiment demonstrates the efficiency and scalability of the generation algorithm and DLKM-based model along several dimensions.

The paper is organized as follows. In Section II, we provide a brief overview of proposed access control models. Then, in Section III, we detail the design of the DLKM scheme including how the ACP technique and Atallah's scheme are seamlessly combined to provide a comprehensive access control model. Next, in Section IV, we design a user-friendly model generation algorithm which serves to democratize the comprehensive and fine-grained DLKM-based CHAC model. In Section V, we perform experiments to illustrate the scalability and efficiency of the proposed model generation algorithm, as well as the DLKM scheme itself, along several dimensions. Finally, in Section VI, we offer concluding remarks.

II. Related Work

Access control of sensitive data is an important and well-studied issue. Over the years, several access control models have been proposed, accepted and deployed in various domains [26], [32].

Discretionary Access Control (DAC) models [10], [28], where the access privileges of each data object is left to the discretion of its corresponding owner, provided flexibility and fine-granularity. DAC models often involved the use of Access Control Lists (ACL) in which individual user-to-privilege mappings could be defined with respect to each data object. Unfortunately, DAC models are not scalable. Defining the privileges of every user with respect to every data object becomes complex and cumbersome as the number of users within an organization grows.

Mandatory Access Control (MAC) models [15], [25], which were widely adopted in military and government domains, allowed a central authority to define organization-wide policies to control data and operation access. MAC models introduced the use of privilege levels when defining access control policies. Within a MAC model, each user and data object could be assigned a corresponding privilege level. Then, users could access data objects of equal or lower privilege level. As a result, MAC models provided additional security by centralizing the definition of access control policies, but the use of privilege levels limited the granularity and flexibility of access controls.

In order to address the weaknesses of DAC and MAC models, while at the same time leveraging their strengths, Role-based Access Control (RBAC) models [11], [27] were proposed. RBAC models organize users within an organization into groups based on their role or activities within the organization. Each role is then, in turn, assigned corresponding data access privileges by a central authority. RBAC models simplified access control policy definition and management, while at the same time offering flexibility and fine-granularity. RBAC models were found to be well-suited for many types of organizations as grouping personnel into hierarchical roles is common within many domains. As a result, the RBAC model has been the most widely accepted and deployed access control model [26].

Since the introduction of RBAC models, several advanced access control models, such as Relation-based Access Control (ReBAC) models [12], [13] and Attribute-based Access Control (ABAC) models [14], [16], have been proposed. Unfortunately, the acceptance and adoption of these recently proposed models is challenged by their complexity.

Unfortunately, none of the aforementioned access control models involve the use of cryptographic keys to enforce access control policies nor do they consider data or user privacy. Therefore, Cryptography-based Hierarchical Access Control (CHAC) models [3] have been proposed. CHAC models typically involve key management schemes which organize users into hierarchical groups. A central authority, known as a group-controller (GC) is often tasked with designing and maintaining the hierarchical structure. Each group is given a corresponding cryptographic key by the GC which can later be used to decrypt and access the data for which the group has been given access privilege. Furthermore, through the hierarchical structure defined by the key management scheme, groups with many access privileges are often able to derive the cryptographic keys of lesser groups in order to successfully access data assigned to the lesser groups. These hierarchical structures leveraged by CHAC models are often used to model and facilitate cryptographically-enforced hierarchical RBAC models.

Many of the early CHAC models had efficiency and flexibility issues. Early CHAC models, such as [3], involved division of two large primes which becomes computationally expensive as the number of bits in the primes increases. Other early models constrained the design of the hierarchy to tree structures or did not allow for efficient modification operations [20], [29], [30]. Later models offered improved flexibility and efficiency of some modification operations [7], [8], [21], [34], but still suffered some inefficient hierarchy modification operations which required re-computation and re-distribution of many groups' cryptographic keys. One elegant scheme, Atallah's Dynamic and Efficient (Extended) Key Management scheme [5], 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.

Although CHAC models directly address the issue of cryptographically enforcing access control policies and data privacy, they also introduce important issues such as key management and distribution. Secure group communication techniques have been proposed to address how to handle key synchronization among users in the same group within a hierarchical access control model. Many different protocols

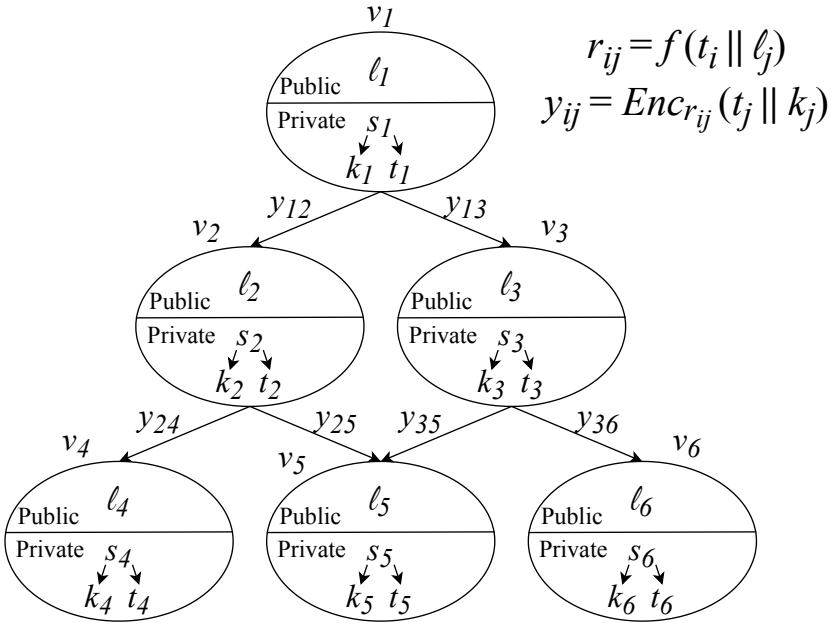

Fig. 1. Example hierarchical RBAC model built using the ACP technique and Atallah's scheme.

have been proposed, including: distributed group key distribution [2], distributed contributory group key agreement [6], decentralized group key management [22] and centralized group key distribution [19]. One interesting solution, the Access Control Polynomial (ACP) [35], 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 formally define the DLKM-based CHAC model [24], [36] generated by our proposed model generation algorithm. The model uses the collusion-resistant and privacy preserving ACP technique [35] in order to distribute a shared secret to a group of users. Such user groups are then organized into a cryptographically-enforced hierarchical RBAC model through the use of Atallah's scheme [5]. The resulting DLKM-based CHAC model provides fine-grained access controls, data and user privacy, and scalability through efficient modification operations.

Like many other CHAC models [3], [7], [8], [20], [21], [29], [30], [34], the DLKM-based CHAC model involves a trusted group-controller (GC). The GC is responsible for designing and maintaining the CHAC hierarchy. Furthermore, the GC is responsible for organizing users into role-based groups and distributing needed shared secrets. As the trusted GC will distribute shared secrets, build the hierarchy and perform modification upon the hierarchy, the GC is be able to derive any cryptographic key used within the model and access any corresponding privileged data.

A. Access Control Polynomials

After the GC has organized a set of users into a role-based group, the GC must distribute a shared secret to each of the group members. To carry out this task, the GC uses the ACP technique [35]. The ACP technique involves a pair of polynomials, a private polynomial $A(x)$ and a public polynomial $P(x)$, each of which is computed over a finite field, F_q, where q is a large γ-bit prime.

The GC begins by computing the group's private polynomial, $A(x)$. In order to compute $A(x)$, the GC begins by obtaining the secret id, $SID_i \in F_q$, of each user in the role-based group, $\psi_i \in \Psi$. A user's SID_i could be derived by information stored by the GC during enrollment, e.g. the user's password hash. Using the SID_i of each user in the group, the GC carries out computation of $A(x)$ as follows:

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

where $||$ denotes a concatenation operation, $z \in F_q$ is a random public-value the GC re-determines during each private polynomial computation and $f : \{0,1\}^* \rightarrow \{0,1\}^\gamma$ is a public cryptographic hash function (i.e. SHA-256 [4]).

Clearly, the private polynomial $A(x)$ computation yields an n-degree polynomial where n is the number of group members assigned to the role-based group, i.e. $|\Psi| = n$. Furthermore, it can be seen that each hash of a secret id concatenated by the public z value, $f(SID_i||z)$, will be a root of the resulting $A(x)$ polynomial. This means that evaluating $A(x)$ with any such value will yield 0, i.e. $A(f(SID_i||z)) = 0$. Using the n-degree polynomial, $A(x)$, the GC can proceed to compute the public polynomial $P(x) \in F_q$ as follows:

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

where $s \in F_q$ is a secret γ-bit value the GC wishes to distribute to each member of role-based group, Ψ. Clearly, the secret

value s will be mixed with and masked by the constant term of $A(x)$. Therefore, without knowledge of $A(x)$ or any of its components, the shared secret s cannot be recovered from $P(x)$.

Note that each group member, $\psi_i \in \Psi$, has access to their own secret id, SID_i, as well as public value z and public cryptographic hash function f. Therefore, any group member can compute $f(SID_i||z)$. By plugging the resulting value into $P(x)$, the group member can recover s as follows:

$$P(f(SID_i||z)) = A(f(SID_i||z)) + s = 0 + s = s \bmod q \quad (3)$$

Therefore, the GC can securely distribute s to each group member, $\psi_i \in \Psi$, by multi-casting $(P(x), z)$. Each time the GC re-computes and re-distributes $(P(x), z)$, they will select a new public random value, $z^{'} \in F_q$. The altered value, $z^{'}$, will serve to dynamically update the two polynomials, i.e. $A^{'}(x)$ and $P^{'}(x)$. The GC can then simply distribute the updated values $(P^{'}(x), z^{'})$.

It should be noted that, through using the above formulation of the ACP polynomials, the public polynomial is attack resistant to adversaries without knowledge of the private polynomial or its components. Fortunately, the above formulation is also resistant to insider collusion attacks. Since any group member can derive s as previously mentioned, the insider attack we consider here is the attack where a subset of a group, $\Phi \subset \Psi$, collude in order to try to derive the secret id, SID_i, of some target user also apart of the group, $\psi_i \notin \Phi$. Since any member of the colluding group can derive s, they can obtain the private polynomial by simply computing $A(x) = P(x) - s$. Then, using any root-finding algorithm, the colluding group may find the root values of private polynomial $A(x)$. As previously mentioned, the roots of the private polynomial $A(x)$ are the hashes of the group members' secret ids, SID_i, concatenated with the public z value, $f(SID_i||z)$. Therefore, if the finite field F_q is sufficiently large and a secure hash cryptographic function, f, is used, it is computationally infeasible for a colluding group to derive any SID_i (regardless of the number of colluding group members).

The collusion-resistant and privacy preserving ACP technique can be further strengthened by the inclusion of dummy roots, $VID_j \in F_q$ in the computation of private polynomial $A(x)$ as follows:

$$A(x) = \prod_{i \in \Psi} (x - f(SID_i||z)) * \prod_{j=1...d} (x - VID_j) \quad (4)$$

where d is the number of dummy roots to be added to the private polynomial and each dummy root VID_j is a value randomly selected by the GC. By adding such dummy roots, insider collusion attacks can be even further defended against. Even in the most adversarial case where the colluding group includes all group members besides a single target group member, $\Phi = \Psi - \psi_i$, it will be impossible for the colluding group Φ to discern if roots of $A(x)$ correspond to the hash of the target user's secret value, $f(SID_i||z)$, or one of the dummy roots VID_j. Therefore, the ACP technique provides a robustly secure method to distribute a shared secret s to a group of users.

Algorithm 1: Atallah's scheme algorithm for 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
```

B. Atallah's Scheme

After organizing users into role-based groups and using the ACP to distribute shared secrets, the GC then uses Atallah's Efficient Key Management scheme [5] in order to organize user groups into a cryptographically-enforced hierarchical RBAC model. An example hierarchy built using the ACP technique and Atallah's scheme can be viewed in Fig. 1.

Atallah's scheme uses a directed acyclic graph (DAG) to represent the hierarchical relationships among roles in the system. The DAG is defined as $G = (V, E, O)$, where V is a set of vertices of cardinality $|V| = n$, E is a set of edges of cardinality $|E| = m$ and O is a set of data objects of cardinality $|O| = p$. A public hash function $f : \{0,1\}^* \rightarrow \{0,1\}^\gamma$, i.e. SHA-256 [4], and symmetric encryption scheme ε, i.e. AES [9], are needed for the derivation and use of symmetric cryptographic keys. Symmetric encryption scheme ε is made up of polynomial-time encryption function $Enc_{SK} : m_i \rightarrow c_i$ and decryption function $Dec_{SK} : \hat{c_i} \rightarrow \hat{m_i}$ where SK is an input encryption/decryption cryptographic key, m_i is a plaintext message and c_i is the ciphertext encryption of m_i.

Each vertex $v_i \in V$ represents a role in the organization, associated with a corresponding group of users, Ψ, and privileges. The GC assigns each v_i a random public label, $l_i \in F_q$, and a random private secret s_i which is shared with the group members, $\psi_j \in \Psi$, via the previously described ACP technique. User ψ_j can derive the two private keys belonging to her group v_i once having s_i: $k_i = f(s_i||0||l_i)$, where $k_i \in F_q$ is used for data encryption and decryption, and $t_i = f(s_i||1||l_i)$, where $t_i \in F_q$ is used for derivation of other groups' private keys.

TABLE I
TIME COMPLEXITY OF KEY DERIVATION AND MODIFICATION OPERATIONS

Operation	Time Complexity
Key Derivation	$O(n)$
Insertion of a Node	$O(1)$
Insertion of an Edge	$O(1)$
User Acceptance	$O(k^2)$

Each object $o_i \in O$ represents a data object belonging to an organization which requires certain privileges in order to access. The mapping function $\mathcal{O} : V \rightarrow 2^O$ maps a vertex to a corresponding set of objects for which it is granted access such that $|\mathcal{O}(v_i)| \geq 0$. One constraint is $\forall i \forall j, \mathcal{O}(v_i) \cap \mathcal{O}(v_j) = \emptyset$ if and only if $i \neq j$, which will enforce each data object $o_j \in \mathcal{O}(v_i)$ is encrypted using a single, corresponding encryption and decryption key k_i. This constraint may lead to hierarchy design and modification difficulties, but our proposed model generation algorithm will later fully address these potential difficulties.

Each directed edge $(v_i, v_j) \in E$ denotes hierarchical relationships between user groups v_i to v_j. In such an edge, v_j is a direct successor of v_i and, likewise, v_i is a direct predecessor of v_j. We also denote such hierarchical relationships as $v_j \in Succ(v_i, G)$ and $v_i \in Pred(v_j, G)$. Furthermore, we denote the entire set of descendants of v_i as $Desc(v_i, G)$. Likewise, we denote the entire set of ancestors of v_i as $Anc(v_i, G)$. It should be note that we consider $v_i \in Desc(v_i, G)$ and $v_i \in Anc(v_i, G)$. Each edge, (v_i, v_j), has a private value, $r_{ij} = f(t_i||l_j)$, and a public label, $y_{ij} = Enc_{r_{ij}}(t_j||k_j)$. The formulation of these edge values enables group v_i to be able to derive the private keys of its successor, v_j (k_j and t_j). Notice that the directed edge formulation does not allow the key derivation of predecessors to be performed by successors. Hence v_i can access the keys of v_j where $v_j \in Desc(v_i, G)$ and, as a result, v_i is granted the access privileges of its descendants, $\mathcal{O}(v_j)$.

The key derivation algorithm of Atallah's scheme [5], $Derive(v_{source}, v_{target}, G)$, is shown in Algo. 1. Using this algorithm, any user in group v_i is able to use her own group's t_i key, along with public node and edge labels, in order to derive the private keys of any descendants $v_j \in Desc(v_i, G)$.

C. Efficient Hierarchy Modification Operations

Atallah's scheme is a very flexible and dynamic key management scheme that supports various kinds of efficient modification operations upon the hierarchy [5]. Each of the modification operations are carried out by the trusted GC. In the context of our proposed user-friendly system (which leverages a user-friendly model generation algorithm), many of these efficient modification operations become fully transparent to the GC. Model generation, as well as many hierarchy modification operations, can be simply carried by the GC inputting a user-friendly representation of the desired access control policies and re-creating the entire hierarchical DAG model using the proposed model generation algorithm which we will see in the next section.

Here, for clarity and completeness, we will briefly discuss how the GC could manually carry out the modification operations used in the model generation algorithm. For an extensive summary of how to carry out every type of possible modification operation, please see [24]. The complexity of the key derivation (shown in Algo. 1) and each discussed hierarchy modification operation can be seen in Table I.

Insertion of a node. When inserting a new node, v_i, to the graph, v_i is first considered as a node without any connections to or from it. The GC simply 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 $t_i = f(s_i||1||l_i)$.

Insertion of an edge. Suppose the directed edge (v_i, v_j) must be added to the graph. 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). In this way, the new edge can support the hierarchical cryptographic key derivations shown in Algo. 1.

Acceptance of a new user. Suppose a new user with secret value SID_{new} is going to be assigned to group v_i by the GC. No group-hierarchy level operations will need to be performed. Instead, the GC only recomputes v_i's ACP with new value $z' \in F_q$ and updated group, $\Psi' = \Psi + \psi_{new}$, as follows:

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

IV. USER-FRIENDLY MODEL GENERATION

In this section we design a user-friendly model generation algorithm (shown in Algo. 2). The algorithm is designed such that the GC is able create and modify the CHAC model's role-based hierarchy without knowledge of its structure or constraints. The GC only needs to provide a list of roles and all corresponding privileges of the particular roles in a user-friendly list-based representation. The algorithm will then automatically generate the corresponding CHAC model which cryptographically-enforces and satisfies the specified access control policies. The overall workflow of model generation is shown in Fig. 2. The GC will then only be responsible for carrying out encryption of sensitive data and distributing the generated group secrets using the ACP technique. As a result, the design and usage of the DLKM-based CHAC model is democratized. For discussion of several specific data encryption and decryption strategies which can be leveraged with the resulting model, please see [24].

A. User Friendly Interface

We begin by creating a user-friendly interface which will be used by the GC in order to input a highly-simplified, list-based representation of access control policies. The goal of

Algorithm 2: Model generation algorithm.

```
1  Create (V*, O*)
2      Initialize V, O
3      for i ∈ (0, length(V)) do
4          for j ∈ (i + 1, length(V)) do
5              if O(v_i) ∩ O(v_j) ≠ ∅ then
6                  V.append(v_length(V))
7                  O(v_length(V)) = (O(v_i) ∩ O(v_j))
8              end
9          end
10     end
11     Sort(O, key = length(O(v_i)), order =
         descending)
12     Initialize P
13     Connect(O, E, P, v_0)
14     return O, G
15 Connect (O, E, P, v_i)
16     if i = length(O) or P_{v_i} ≠ ∅ then
17         return
18     end
19     for j ∈ (i + 1, length(O) do
20         Connect(O, G, P, v_j)
21         if P(v_j) ∈ O(v_i) and P(v_j) ∉ P(v_i) then
22             E.append(v_i, v_j)
23             P(v_i) = P(v_i) ∪ P(v_j)
24         end
25     end
26     O(v_i) = O(v_i) − P(v_i)
27     P(v_i) = O(v_i) + P(v_i)
28     return
```

this interface is to provide the user a simple and easy way to implement a CHAC. As the described hierarchical access control model is subject to design constraints, a GC without knowledge of symmetric encryption, key management and hierarchical access control models, is likely to struggle to successfully design the correct DAG hierarchy corresponding to their desired access control policies. Albeit, even if the GC has the necessary background knowledge of CHAC models and the data structures involved, they will likely still encounter difficulties. A hierarchical model with even a relatively small number of roles quickly becomes tedious and difficult to design by hand.

Each role in the hierarchy is assumed to have a certain number of associated objects which it has the privilege to access. In a RBAC model the user would simply define the privileges for each role, however, when using a cryptographically-enforced CHAC model, there are additional concerns. In a CHAC model, roles are arranged as nodes in a DAG where each node is able to access privileged data assigned to itself or any of its children. Instead of simply defining privileges for each role, the designer must define roles is such a way that shared privileges are gained by greater roles through traversing down the hierarchy to lesser, descendant roles. Designing a hierarchy based on numerous role-to-privilege mappings is unpleasant at best and infeasible at worst.

Therefore, we use a web-based application to provide the GC a user-friendly interface. This web application utilizes the Python framework, Django, to implement the server-side code and Javascript to implement the client-side code. The interface allows the GC to add/remove possible privileges, add/remove roles, and to assign a possible privilege to a particular role. Once a role is created, the GC is able to easily mark the privileges they want this role to have. From the GC's perspective, she is able to define privileges in same intuitive manner as she would using an RBAC scheme. However, transparent to the GC, our model generation algorithm will arrange the roles into a cryptographically-enforced hierarchy where each role can derive all its assigned privileges. After generating the CHAC, the interface will use a Javascript graph visualization library to display the resulting DAG to the user. An example screenshot of the proposed user interface is shown to the left in Fig. 2. Furthermore, the output visualization of a generated model is shown to the right.

B. Model Generation Algorithm

Now, using the GC's input through the aforementioned user-friendly interface, the system can carry out model generation. The model generation is carried out through Algo. 2. Here we describe the algorithm in detail.

1) (Algo. 2, Line 2) First, we begin by initializing G with the given input, V^* and $\mathcal{O}^*$, from the user-friendly interface. Recall that the user-friendly input, V^* and $\mathcal{O}^*$, will be a list-based representation of roles and corresponding privileges, so the Attalah scheme constraint, $\forall i \forall j, \mathcal{O}(v_i) \cap \mathcal{O}(v_j) = \emptyset$, will likely not be satisfied. Using the user-friendly input, we create a vertex for each given role, $V = set(v_i \in V^*)$, and begin by assigning all corresponding privileges to each of the roles according the given access control policies, $\mathcal{O} = \mathcal{O}^*$.
2) (Algo. 2, Line 3–10) As mentioned in the previous step, one important constraint of Atallah's key management scheme is that there should be no intersection of privileges directly assigned to any two roles. This is because only one symmetric data encryption and decryption key should be used to access the data. In order to satisfy this constraint, we introduce extra placeholder roles to hold the possible intersections of privileges. If we let $k = length(V)$, we add v_k as a placeholder in V using the insertion of a new node operation defined in Sec. III-C. We then assign $\mathcal{O}_{v_k} = \mathcal{O}_{v_i} \cap \mathcal{O}_{v_j}$.
3) (Algo. 2, Line 11) Next, we sort the role-privilege list, $\mathcal{O}$, such that roles are ordered by the amount of privileges that have in descending order. This sorted ordering will ensure correctness when generating edges to form the hierarchy.
4) (Algo. 2, Line 12–13) Edges are then generated in order to build the hierarchy and make certain that each privilege is only assigned to a single corresponding role

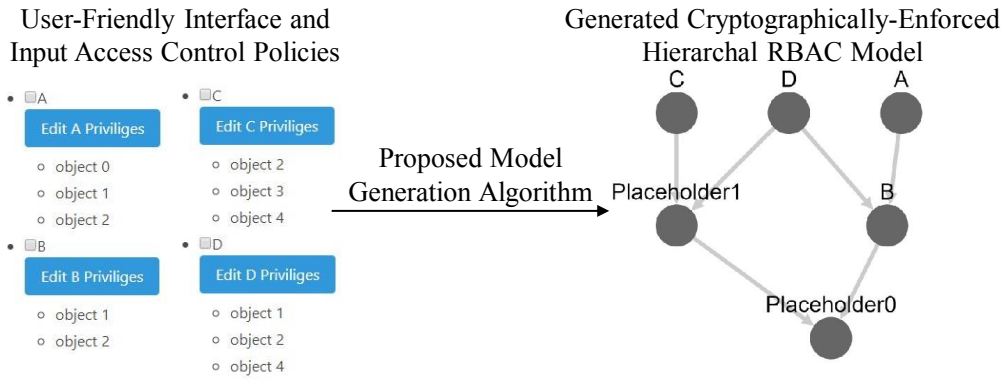

Fig. 2. Model generation using proposed Algo. 2. Left: The designed user-friendly interface described in Sec. IV-A. As shown, the interface is used by the GC to input easily-understandable, list-based access control polices. Right: The generated cryptographically-enforced hierarchical RBAC model corresponding to the input access control policies.

in the model. To accomplish this, we need to call a recursive helper function to traverse the role-to-privilege mapping, $\mathcal{O}$. P is initialized to serve as an additional mapping of roles to corresponding privileges. P is used in the recursive function to record the privileges each role can access from all its descendants. Therefore, $P(v_i)$ will have all the privilege accessible to v_i when the helper function returns.

5) (Algo. 2, Line 16–18) These lines define the base case of the recursive helper function. If we reach the end of the $\mathcal{O}$ list or if v_i was visited before, we can return from the recursive function.
6) (Algo. 2, Line 19–25) Here we traverse all the roles and add edges from each role to its successors. As we sorted $\mathcal{O}$, successors of role v_i will only appear after it in the list. We can tell if v_j is a direct successor of v_i if $P(v_j) \subset \mathcal{O}(v_i)$ and if there is no $P(v_j) \subset P(v_k)$ where v_k is already a descendant of v_i. If v_j is considered to be a successor of v_i, we add an edge (v_i, v_j) to E. This edge insertion operation is performed using the steps described in Sec. III-C. Finally, we update $P(v_i)$ to be the union of itself and $P(v_j)$. This is because v_i can now inherit all privileges of v_j through derivation of the keys of v_j using the newly inserted edge (v_i, v_j).
7) (Algo. 2, Line 26–27) Finally we have the graph information for v_i needed to facilitate the access privileges assigned to v_i by the GC. Furthermore, the generated graph information will satisfy the uniqueness requirement of privilege assignments. We then need to eliminate the privileges accessed from $Desc(v_i, G)$ from $\mathcal{O}(v_i)$. We finally record all inherited privileges of v_i in $P(v_i)$ before we return.
8) (Algo. 2, Line 14) Finally, $\mathcal{O}$ is the role-privilege mapping generated for CHAC model and G contains the graph information that the GC can leverage for future use.

V. Experiment

This section details the computational and memory scalability of the resulting system. Each experiment was ran on a modest Lenovo Thinkpad 13 Ultrabook laptop with a Intel i5-6200U CPU and 8GB of RAM.

A. Model Generation Experiment

It is important to note that this interface is intended for real-world use in which algorithm run-time is of great concern. Therefore, this experiment illustrates the speed the model generation algorithm as the hierarchy DAG increases in size. The model generation algorithm is first tasked with creating a 10 role graph. With each subsequent trial, the number of roles is then incremented by 10 until reaching a graph of 100 roles. Each role is then assigned 10 random privileges from a set of possible privileges. The set of possible privileges will be 10 times the number of roles used. For example, in a graph with 10 roles there will be a total of 100 possible privileges where each role will randomly be assigned 10 privileges.

After generating roles and randomly assigning privileges, the algorithm will generate a DAG for the model. It is important to note that the number of roles input may not be the same as the number of nodes produced in the resulting graph. As discussed in the model generation section, there may be cases in which a placeholder node must be introduced to maintain the hierarchical structure. Also note that the experiment only measures the run-time of the model generation algorithm and not any time taken to define the role/privilege mapping.

For each input size, the experiment was ran 10 separate times with a different random role/privilege mapping each time. Since program run-times may naturally fluctuate, the average of 10 run-times is what is reported.

The results of the experiment can be seen in Fig. 3. The x-axis represents the number of roles used while the y-axis represents the amount of time in seconds to complete the model generation algorithm. It can be seen that run-time increases linearly with the number of roles. At the highest point on the graph the run-time is seen to be as much as 16

seconds. The input size at this point is 100 roles with a total of 1000 possible privileges where the DAG created may have to create many different placeholder nodes in order to maintain its properties. The resulting DAG is highly complex and would be extremely difficult to design by hand. With this in mind, while the user may have to wait a few seconds, they will still be saving a great deal of time.

Due to the random nature of assigning privileges, it is likely that in practice user-defined structures would be less intensive to generate. Also, if a real world application needs to generate such a complex DAG it is likely that a more powerful computer will be used. This experiment provides strong indication that the system is scalable and can democratize complex model generation.

B. Memory Consumption Experiment

This experiment provided analysis of memory consumption by the system. Again, the x-axis represents the number of roles used in the randomly generated role/privilege mapping. The y-axis represents the amount of kilobytes used by the DAG object created by the model generation algorithm.

The results of the experiment can be see in Fig. 4. Memory consumption clearly grows linearly with respect to the number of roles. This is because each node added to graph requires the same amount of storage space. The increase in memory consumption varies slightly due to the fact that placeholder nodes may increase the number of nodes to a value greater than the number of roles. However, the increase in memory consumption is negligible. At the largest DAG size of 100 roles and 1000 objects the required memory consumption is less than 100KB. This small memory consumption strongly supports the scalability of the proposed system.

C. Key Derivation Experiment

The final experiment examined the scalability of the key derivation algorithm in conjunction with the DAG created by the model generation algorithm. In this experiment privileges are no longer assigned randomly. Instead privileges are assigned in such a manner that the DAG created will be composed of a single directed chain of nodes. For example, the root level node has all possible privileges assigned to it. Then its direct successor node has all of the same privileges except for one. The privileges of each node is decremented in this manner until the final leaf node only has a single privilege. This ensures that trials with the same number of roles as an input will have the same structure.

Once more, the x-axis represents the number of nodes along the path that must be traversed using Algo. 1 in order for the root node to derive the leaf node's keys. The y-axis represents the amount of time in seconds to derive the key of the leaf node starting at the root level node. Overall, the key derivation run-time increases linearly with respect to the number of roles. The small variation in jumps between input sizes is mostly due to the inconsistent nature of run-time measurement. At some points, run-times seems to increase non-linearly, but this is simply due to the scale of the y-axis. The algorithm is fast enough that run-time inconsistencies appear to result in sharp increases, but overall the algorithm execution time is still quite short. With 100 roles the key derivation algorithm is still able to derive the key of the leaf node in under 0.03 seconds. This quick key derivation is critical as users will expect key derivation, data decryption and data access to happen in near real-time speeds. The experimental results strongly indicate the system is scalable to and can provide quick data access.

VI. Conclusion

In this paper, we have presented a user-friendly model generation algorithm which can effectively generate a DLKM-based CHAC model. The generated model is able to facilitate a cryptographically-enforced hierarchical RBAC model. In addition to the security and privacy benefits of the generated model, our experiments demonstrate that it is highly scalable and efficient. As the proposed model and generation algorithm are scalable, privacy-preserving and user-friendly, there is clear motivation for their adoption by large organizations which leverage sensitive data. Our Python code implementation is provided for open use at [1].

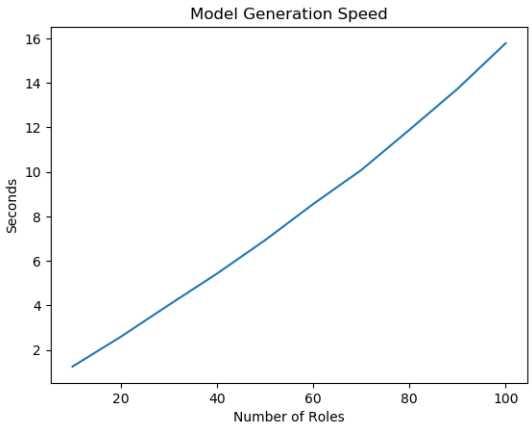

Fig. 3. Model generation algorithm speed.

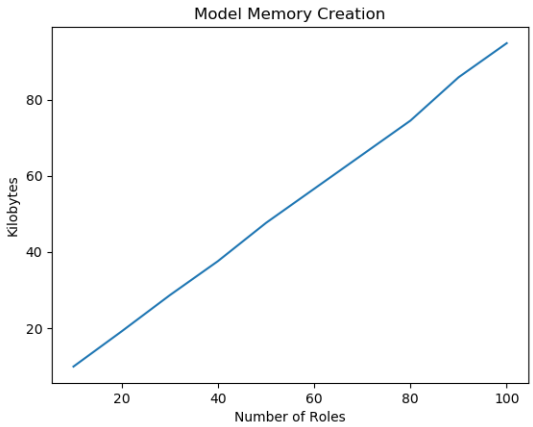

Fig. 4. Model memory consumption.

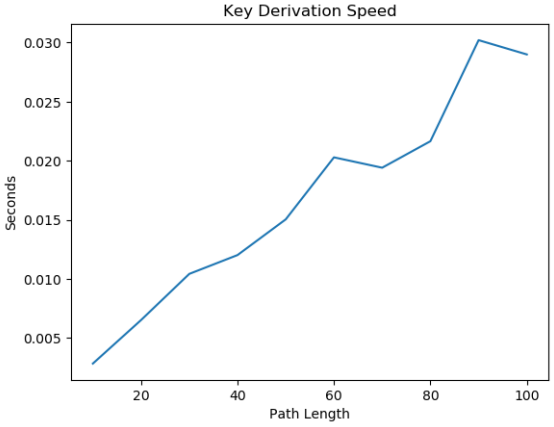

Fig. 5. Key derivation speed.

As future work, we plan to expand the proposed interface and model generation algorithm. As we allow an inexperienced GC design and modify the hierarchy from the user-friendly list-based access control policy perspective, we would also like to allow an experienced GC to be able to directly modify the generated DAG model. Such an extended interface would allow the GC to modify the model from two perspectives, giving them enhanced control of the model they adopt and deploy for their organization.

ACKNOWLEDGMENT

This work is partially supported by a U.S. National Science Foundation grant (CICI #1839746) and also the NSF Jetstream [31]/XSEDE [33] 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/haakensonb/tool_interface*.
[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] Andrew W. Appel. Verification of a cryptographic primitive: Sha-256. *ACM Trans. Program. Lang. Syst.*, 37(2), April 2015.
[5] 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.
[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] R. C. Joseph. Data breaches: Public sector perspectives. *IT Professional*, 20(4):57–64, Jul 2018.
[18] HIPPA Journal. Hipaa encryption requirements. *https://www.hipaajournal.com/hipaa-encryption-requirements/*.
[19] Xiaozhou Steve Li, Yang Richard Yang, Mohamed G Gouda, and Simon S Lam. Batch rekeying for secure group communications. *group*, 1:9, 2001.
[20] 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.
[21] Chu-Hsing Lin. Hierarchical key assignment without public-key cryptography. *Computers & Security*, 20(7):612–619, 2001.
[22] Suvo Mittra. Iolus: A framework for scalable secure multicasting. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 277–288. ACM, 1997.
[23] National Institute of Standards and Technology. Guideline for using cryptographic standards in the federal government. *https://www.nist.gov/news-events/news/2019/07/guideline-using-cryptographic-standards-federal-government-cryptographic*.
[24] T. Phillips, X. Yu, B. Haakenson, and X. Zou. Design and implementation of privacy-preserving, flexible and scalable role-based hierarchical access control. In *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 46–55, Dec 2019.
[25] Patrick Sack, Edward Austin, and Scott Gaetjen. Mandatory access control label security, November 9 2010. US Patent 7,831,570.
[26] E. Sahafizadeh and S. Parsa. Survey on access control models. In *2010 2nd International Conference on Future Computer and Communication*, volume 1, pages V1–1–V1–3, May 2010.
[27] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.
[28] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.
[29] 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.
[30] Ravinderpal S Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27(2):95–98, 1988.
[31] 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.
[32] Vivy Suhendra. A survey on access control deployment. In *FGIT-SecTech*, 2011.
[33] J. Towns and T. Cockerill et al. XSEDE: Accelerating scientific discovery. *Computing in Science & Engineering*, 16(5):62–74.
[34] Sheng Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8):750–759, 2002.
[35] 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.
[36] 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.