Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 23 (2007) 776–786
FGCS
FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Dual-Level Key Management for secure grid communication in dynamic and hierarchical groups

Xukai Zou, Yuan-Shun Dai*, Xiang Ran

Department of Computer and Information Science, Purdue University School of Science, Indiana University, Purdue University, Indianapolis, 46202, USA

Received 31 March 2006; received in revised form 26 September 2006; accepted 11 December 2006
Available online 22 December 2006

Abstract

Grid computing is a newly developed technology for complex systems with large-scale resource sharing and multi-institutional collaboration. The prominent feature of grid computing is the collaboration of multiple entities to perform collaborative tasks that rely on two fundamental functions: communication and resource sharing. Since the Internet is not security-oriented by design, there exist various attacks, in particular malicious internal and external users. Securing grid communication and controlling access to shared resources in a fine-tuned manner are important issues for grid services. This paper proposes an elegant Dual-Level Key Management (DLKM) mechanism using an innovative concept/construction of Access Control Polynomial (ACP) and one-way functions. The first level provides a flexible and secure group communication technology while the second level offers hierarchical access control. Complexity analysis and Simulation demonstrate the efficiency and effectiveness of the proposed DLKM in both computational grid and data grid. An example is illustrated.

Keywords: Grid computing; Grid security; Group communication; Hierarchical access control; Key management

1. Introduction

Grid computing [14] is a recently emerging technology focusing on large-scale resource sharing and multi-institutional collaboration, see e.g. [19,11,17,15,3]. One of the most important issues in the Grid is security [20,25]. The Internet and networks are not security-oriented by design. Numerous hackers constantly explore security holes existing in hardware, software, processes, or systems to launch various attacks. There are two types of attacks: passive and active [4,18,27]. Passive attackers steal useful information by eavesdropping and/or performing traffic analysis. Active attacks interfere with legal communication and are typically in the forms of masquerading, replaying, modification, and denial of services (DOS). The countermeasures against attacks utilize encryption/decryption for confidentiality, message authentication code for integrity, digital signature for authentication, undeniable digital signature for non-repudiation, access control for authorization, and intrusion detection/defence for availability/DOS [24]. Most of these technologies are based on cryptography in which keys and key management are the most important but complicated issues.

The Internet-based grid computing encounters the same attacks and involves all the security requirements discussed above. Furthermore, grid computing systems are group-oriented, including a large number of users and shared resources. They are also complex, dynamic, distributed and heterogeneous. As a result, the attacks to grid systems may become more serious and to defend them becomes more difficult. For example, due to the distributed and heterogeneous features of grid computing systems, centralized authentication is generally unavailable and multiple-site co-authentication is difficult to implement. Thus, the Single-Sign-On [16] authentication comes into play. For another example, grid computing is aimed at providing collaborative services. These services are featured by two important functions: group-oriented communication and information sharing/exchange [30]. As long as communication and information exchange are conducted over the Internet, communication messages should be encrypted with a common key for confidentiality. However,

Abbreviations: KMS: Key Management Server, DLKM: Dual-Level Key Management, CA: Central Authenticator, RMS: Resource Management System, ACP: Access Control Polynomial, HAC: Hierarchical Access Control.

* Corresponding author. Tel.: +1 317 274 3473.
E-mail address: ydai@cs.iupui.edu (Y.-S. Dai).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.12.004

Notations

SID_i	Every valid user/machine is assigned a permanent secret key SID_i
z	A random integer which is changed and made public every time
$A(x)$	Access Control Polynomial (ACP)
$P(x)$	Public polynomial sent to users for key distribution, $P(x) = A(x) + K$
P	The system prime used for modular computation
U_i	A group member in a certain group
v_j	A certain vertex in the second level hierarchy
$\hat{k}_i$	A secret group key
k_i	A private group key
$f(x, y)$	The public one-way hash function
ID_i	A unique public identity assigned to each vertex in the second level hierarchy
m	The number of users in a certain node
n	The number of vertices in the hierarchy structure
$p_{i,j}$	The public edge value on the edge from v_i to v_j
%	Modular operation

due to the high dynamic nature of grid computing, how to update group key(s) efficiently and effectively becomes a challenging problem. As for resources sharing among different nodes/organizations in the grid, every participating node would like to offer its resources to be used by other nodes. However this sharing must be in a controllable and fine-tuned manner. Thus, security is of great concern to grid computing and the solutions for different security problems need to be studied and designed in a holistic manner.

To solve the above critical security problems that set obstacle for the further deployment and applications of grid computing, this paper presents a novel Dual-Level Key Management (DLKM) scheme that is appropriate to grid context. At the first level, Dynamic Groups and Key Distribution are the focus, where a novel and efficient scheme using Polynomial as public information to hide a group key is presented. These dynamic and independent polynomials enable secret information to be distributed to arbitrary and dynamic user groups. At the same time, they are able to defend against different attacks including collusion of malicious internal users. The second level is aimed at solving Hierarchical Access Control (HAC) and makes use of a new hybrid scheme. The HAC scheme allows a member to derive the key of any of its descendants efficiently, but the reverse is prohibited. Moreover, the first-level is the basis of the second level and helps accurately distribute secret information to any particular node in the second level hierarchy. One important feature with DLKM is that it allows users, processes, and resources to freely enter or leave a grid system without jeopardizing required security. Thus, this technology could help promote grid computing to a new era, in which security-critical services offered on the grid is enabled.

The rest of the paper is organized as follows. Section 2 describes grid computing and presents a novel Dual-Level Key Management (DLKM) System for grid computing and services. Section 3 analyzes security-related measures, robustness and complexity of the proposed DLKM. In Section 4, DLKM is implemented into a grid service, where a numerical example is illustrated and some performance measures from a real grid computing case are depicted. Section 5 briefly discusses the relation of the proposed DLKM with the grid security architecture and with the existing SGC and HAC techniques.

2. Secure grid communication

2.1. Grid computing and security challenges

The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [17]. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources. This is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry and science.

However, there is a significant security challenge on the Grid resource sharing, that is, the data privacy to others outside the group of shared resources. When those resources involved in a task communicate with one another, the task owner may not want other untrusted/unauthorized people to know the communication data. Unfortunately, the Internet contains many malicious factors (such as hackers, virus) especially for the grid. One cannot expect everybody on the Internet to be trustworthy. Thus, the information transmitted among remote sites/resources should be encrypted.

Point-to-point cryptographic schemes have been well developed such as RSA, but Grid computing is featured by collaboration among a group of users and their sharing of computational or data resources. Therefore, it is very inefficient to unicast common information one to another, but multicasting shared information among the group is much more efficient. The multicast information needs to be encrypted (by a group key) so that others cannot understand the information even though they might intercept it. Groups can be dynamic, because users, resources or sites can attend or leave a group at any time, and groups are organized in real-time according to the availability and workload of various resources. In addition, one member may belong to multiple groups simultaneously. Thus, the follow-up challenges emerge as how to authenticate the group members, how to distribute the group key to the group members and how to update the group key securely and efficiently when the group members change.

Another important feature of grid computing is Hierarchical Access Control (HAC). This scenario is mainly related to the management in grid computing. The grid environment consists of resources and users (the number of users and resources can vary from a few to millions). There are different relations among users and resources. Some special members/nodes have been authorized to monitor the tasks of certain resources or to check the communication among some grid resources, such as system administrators, service/resource providers, Resource Management Systems (RMS) and so on. Hence, they should be able to possess the access keys of the resources under their authority while unable to obtain the keys of others out of their authority. Today, a grid can become increasingly large so a hierarchical controlling model is widely deployed, i.e. there are different levels of system administrators, providers, users and RMSs. The lower-level members/sites/nodes are controlled and monitored by the higher-level ones. This needs hierarchical key management.

In order to solve the above security challenges in a holistic manner, we present a new and efficient methodology of Dual-Level Key Management for grid computing as follows.

2.2. *Dual-Level Key Management (DLKM)*

DLKM consists of two levels: the first to solve the problem of Dynamic Groups and Key Distribution; the second designed for HAC, which is built upon the first level.

2.2.1. *First level*

It is assumed that every valid user/machine in the system is assigned a permanent secret key, denoted by SID_i for member U_i (the member here generally represents the user/node/machine in the grid). For example, when a user or an organization registers to the Grid via the Globus Toolkit (http://www.globus.org/toolkit/), several certificates need to be issued, including the host certificate that authenticates the machine involved in the grid, the service certificate that authenticates the services offered to the grid, and the user certificate that authenticates use of the grid services. In this registration process, the permanent secret key can be embedded into the certificates issued to the member. Assume P is a large prime which forms a finite field F_p.

Whenever there will be a group of users participating in a grid service, the Key Management Server (KMS) will construct a polynomial $A(x)$ in finite field $F_p[x]$ as:

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

where ψ denotes this group under consideration and SID_i are group members' permanent secret keys assigned to the members in ψ. $f(x, y)$ is a public one-way hash function and z is a random integer from F_p. $A(x)$ is called an *Access Control Polynomial* (ACP). As Eq. (1), it is apparent that $A(x)$ is equated to 0 when x is substituted with $f(SID_i, z)$ by a valid user with SID_i in ψ; otherwise, $A(x)$ is a random value.

The KMS selects a random group key K for group ψ and computes the polynomial:

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

Finally, the KMS publicizes $(z, P(x))$.

From this public information, any group member U_i can get the key by:

$$K = P(f(SID_i, z)). \tag{3}$$

Here U_i computes $f(SID_i, z)$ first and then substitutes into $P(x)$.

For any other member U_r excluded by ψ, $P(f(SID_r, z))$ yields a random value from which U_r cannot get the hidden key K. This key management mechanism guarantees that only a user whose SID_i is included in $A(x)$ can extract the key from $P(x)$.

With this scheme, dynamic groups can be easily managed to accept and revoke users. If a new user U_t needs to be added, the KMS creates a new SID_t and assigns it to U_t. Then, the KMS includes $(x - f(SID_t, z))$ in the formation of $A(x)$ and gets

$$A'(x) = \prod_{i \in \psi} (x - f(SID_i, z))(x - f(SID_t, z)) \tag{4}$$

$A'(x)$ is used to mask key K by computing $P'(x) = A'(x) + K$. Then $(z, P'(x))$ is sent to U_t. After receiving $(z, P'(x))$, U_t can use SID_t to derive the key from Eq. (3).

If a current group member U_t needs to be revoked from the group, the KMS just selects a new random z' and recomputes $A'(x)$ by excluding the corresponding $(x - f(SID_t, z'))$. Then, the KMS selects a new group key K', computes $P'(x) = A'(x) + K'$, and multicasts $(z', P'(x))$. Now, the deleted user U_t cannot extract K' from $P'(x)$.

2.2.2. *Second level*

The second level is to solve the HAC problem based on the first level. In a hierarchy, a vertex in the higher level is designated to have the access rights associated with the vertices which are the descendants of the vertex. However, the reverse is prohibited. Cryptography-based HAC works as follows. Every node in the hierarchy is assigned a cryptographic key. The vertex in the higher level can derive, from its own key, the keys of its descendant vertices at the lower levels. The reverse is not true. Let us see an example with a typical HAC scheme proposed by Lin [21]. Suppose every node is assigned a public identification and a private node key. Every edge between two nodes is assigned a public edge value computed from a one-way hash function. For example, suppose node v_i with public ID_i and private k_i is a parent of node v_j with public ID_j and private k_j, then the public value on the edge from v_i to v_j will be $p_{i,j} = k_j \oplus f(k_i, ID_j)$ where $f(x, y)$ is a one-way hash function. Thus, v_i can compute v_j's key as $k_j = p_{i,j} \oplus f(k_i, ID_j)$. Moreover, if v_i is an ancestor of v_j, v_i can derive v_j's key iteratively along the path from v_i to v_j. However, v_j cannot reversely derive v_i's key. The advantage of this scheme is that since nodes' keys are independent, the key change of one node will not affect its descendant keys, but will cause update to public edge information.

We will adopt the above Lin's scheme as our second level HAC mechanism (with the following enhancement). One most serious problem with Lin's scheme is that when a member leaves from a node such as v_i, not only v_i's key k_i needs to be changed and distributed to all the remaining users in node v_i, but also the keys of all v_i's descendants need to be changed since the revoked member already knew all these keys. This is obviously a key updating problem. In order to solve this problem, we adopt a mechanism proposed in [2] which in fact adds one more key to any node and uses the one-way hash function $f(x, y)$ one more time. The idea is as follows. Suppose

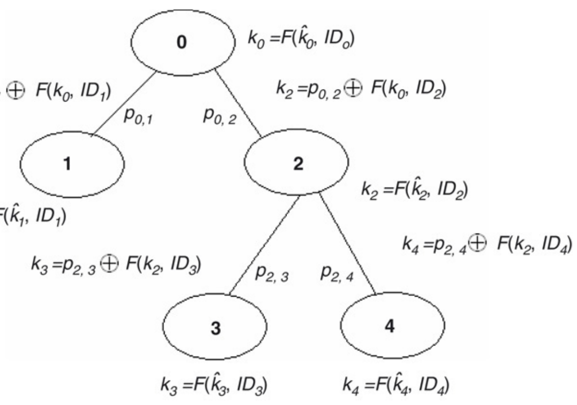

0
$k_0 = F(\hat{k}_0, ID_0)$
$k_1 = p_{0,1} \oplus F(k_0, ID_1)$
$p_{0,1}$
$p_{0,2}$
$k_2 = p_{0,2} \oplus F(k_0, ID_2)$
1
2
$k_2 = F(\hat{k}_2, ID_2)$
$k_1 = F(\hat{k}_1, ID_1)$
$k_4 = p_{2,4} \oplus F(k_2, ID_4)$
$k_3 = p_{2,3} \oplus F(k_2, ID_3)$
$p_{2,3}$
$p_{2,4}$
3
4
$k_3 = F(\hat{k}_3, ID_3)$
$k_4 = F(\hat{k}_4, ID_4)$

Fig. 1. An example of access hierarchy and their key and edge values.

every group node v_i has one more key, i.e. a secret key $\hat{k}_i$, which is distributed to the users of node v_i securely. Then the group private key k_i is defined as $k_i = f(\hat{k}_i, ID_i)$. Since the users in node v_i have the secret key $\hat{k}_i$, and ID_i is public, every user in v_i can compute the private key k_i himself. As before, public edge values are defined on private keys. A parent node can derive the private key k_i of its descendant v_i but not the secret key $\hat{k}_i$. The change of ID_i can alter k_i easily. Hence, only when there are users who leave from v_i, $\hat{k}_i$ needs to be changed and distributed to the users in v_i and k_i needs to be updated from $\hat{k}_i$. All other changes of k_i can be easily done by just changing public ID_i. In summary, our second level HAC scheme is defined as follows:

(i) for every node v_i, it is assigned a secret key $\hat{k}_i$, a public ID_i, and a private key k_i which is computed by itself as:

$$k_i = f(\hat{k}_i, ID_i) \tag{5}$$

(ii) for every edge from v_i to v_j, a public edge value $p_{i,j}$ is defined as:

$$p_{i,j} = k_j \oplus f(k_i, ID_j). \tag{6}$$

Fig. 1 illustrates an access hierarchy, keys and edge values. Each vertex represents a group of members formed from the first level.

Now, the access control and dynamic key update are ready. Let us show different operations.

(i) Key derivation: Assume that vertex v_i is a parent node of vertex v_j, v_i can derive k_j by using its own private key k_i and public information $p_{i,j}$ and ID_j as:

$$k_j = p_{i,j} \oplus f(k_i, ID_j). \tag{7}$$

but vertex v_j cannot compute k_i because of the one-way feature of the function.

(ii) Similarly, if vertex v_i is an ancestor node of vertex v_j, following the path from v_i to v_j, v_i can derive v_j's key iteratively from its own key.

(iii) When a vertex's key needs to be updated, the update does not affect the keys of other vertices but just the edge values of its adjacent vertices. Assume vertex v_i's key k_i needs to be changed. A new ID'_i is picked up from F_p and the new key k'_i is computed as $f(\hat{k}_i, ID'_i)$. Then, for v_i's parent vertex v_t, $p_{t,i} = k'_i \oplus f(k_t, ID'_i)$ is computed as public information. Similarly, the edge values to v_i's child node v_j are updated as $p_{i,j} = k_j \oplus f(k'_i, ID_j)$.

(iv) Adding and removing a vertex can be processed in a similar way. If a new vertex v_i needs to be added, we just pick up a new secret key $\hat{k}_i$, and a new ID_i and assign them to node v_i. Then, we compute and publicize the edge values to its parent vertices and child vertices. Similarly, if an existing vertex v_i needs to be removed, we delete all edges adjacent to vertex v_i and then remove v_i.

(v) Adding/deleting an edge: Adding an edge means computing and publicizing the edge value. As for deleting an edge such as from v_i to v_j, if there exist other paths from v_i to v_j, nothing needs to be done since v_i still can compute v_j's key via other paths. But if this is the only path from v_i to v_j, v_i should not be able to compute v_j's key (and all v_j's descendants' keys) after deletion. This means that v_j's key and all v_j's descendants' keys need to be changed. This change can be done by changing their public *ID*s only without regenerating or resending the secret keys.

2.2.3. *Combination of the two levels*

Now, let's see how the two levels work together. After the two levels are combined, all members in a group represented by vertex v_i share one identical group secret key $\hat{k}_i$ on the second level. Each member U_j has a personal secret key SID_j, as registered at the first level. Every vertex v_i has its own unique group secret key $\hat{k}_i$ and $A_i(x)$. Here $A_i(x)$ is composed out of all *SID*s of the members in v_i and used to distribute $\hat{k}_i$ to all v_i's members via the multicast of $P_i(x)$. Then, all (and only) the members in vertex i can derive the group secret key $\hat{k}_i$. Following that, the group private key k_i can be obtained via Eq. (5) by members themselves.

At the first level, when a member U_r leaves a group (vertex v_i), a new group secret key $\hat{k}'_i$ is distributed by computing $A'_i(x)$ without the term $(x - f(SID_r, z'))$ and multicasting $P'_i(x)$. Then, the remaining group members can derive the new group private key k'_i by Eq. (5) but U_r cannot. The new $\hat{k}'_i$ and k'_i will result in corresponding changes in the second level. Three types of updates will be performed: (1) all v_i's parent edge values; (2) all the public IDs (thus, private node keys) in v_i's sub-tree; and (3) all the edge values in v_i's sub-tree. These renewing steps can effectively prevent the removed user U_r from extracting the new keys of its former descendant vertices.

The advantage of DLKM is obvious by combining the two levels. At the first level, $A(x)$ and $P(x)$ make key distribution efficient and secure. Also, handling group dynamics is simple by just assigning SID_i and/or adding/deleting corresponding terms in $A(x)$. At the second level, edge values computed from a one-way hash function guarantee the higher level vertex's access control over the lower level vertices. Moreover, updating a node's private key can be done easily by changing the node's ID. As a result, their combination makes sure that the change to one vertex does not have influence on other vertices (e.g. their private node keys).

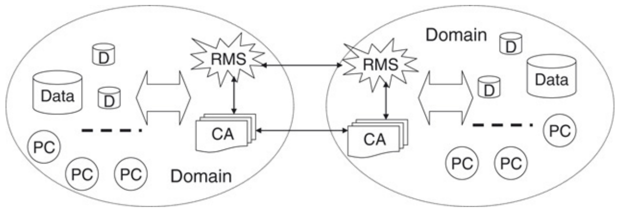


Fig. 2. General architecture of grid computing.

2.3. Secure grid computing by DLKM

The proposed DLKM can make grid services secure in data communication and access control. To implement it into the grid computing, it is important to merge DKLM with the general architecture of grid computing, as depicted by Fig. 2.

In each domain of the grid system, there is a Central Authenticator (CA) which issues the certificates for hosts, users and services when they are first registered to the grid system. The first step of DLKM to distribute personal secret keys can be combined with the process of issuing the certificates from the CA. Since registration is a point-to-point process between the CA and a member, any two-party public/private key scheme like RSA can be used to encrypt the initial information.

The CA can then act as the KMS to distribute keys using DLKM in an efficient and secure manner. There are two typical types of resource access in grid computing: (1) Access Local Resources inside a domain as shown by Fig. 1; (2) Access Remote Resources of other domains. How to secure both cases is described as follows:

2.3.1. Access Local Resources

Based on the above architecture design, accessing local resources can be done as follows (note: the generic process of requesting grid services is also included):

(1) A user submits a request for a certain grid service to the RMS controlling this domain;
(2) The RMS detects a group of available resources (including computational resources and data resources);
(3) The RMS informs the CA. The CA randomly generates a key K_1 and then uses the private key k_i of node v_i to encrypt K_1 where v_i is the vertex these resources belong to:
(4) The CA multicasts the encrypted key to these resources and to the RMS;
(5) The resources with k_i can decrypt K_1. The RMS can derive k_i first. Note: The KMS is usually at the top of the hierarchy and v_i is its descendant. Then, the RMS can also obtain the same K_1;
(6) Thus, these resources together with the RMS can use this common key K_1 to communicate. For example, the RMS needs to send jobs/programs to the resources, then the resources need to communicate/access/exchange data, and finally the results need to return to the RMS for composition. All of these communication processes need the common key K_1 to encrypt communication messages for data privacy;
(7) During the process, if certain members (such as system administrators) need to monitor the whole process, they can derive the private key k_i as long as v_i is their descendant, and then decrypt K_1 from the public information generated by the CA. Therefore, they can monitor the process if they want.

In step 3, we assume those resources belong to one group/vertex. However, those available resources may belong to different groups/vertices. The simple way to solve this problem is that the CA encrypts K_1 multiple times corresponding to the multiple groups/vertices involved, and then multicasts. Again, the join/leave/movement of users among groups has already been described above Section 2.2.

2.3.2. Access Remote Resources

Suppose a user submits a request which need to use not only the resources in the local domain but also the resources of some other remote domains (controlled by RMS_j, $j = 1, 2, \ldots, J$). Each domain has its own CA, so the combined group of local and remote resources can get the key distributed as follows:

(1) A user submits a request for a certain grid service to the local RMS;
(2) The local RMS finds that besides some local resources, the request needs to use remote resources from other domains;
(3) The local CA picks up a random K_1 and distributes K_1 to those local resources and the local RMS (following the same procedure as above);
(4) The local CA talks with other CAs of those remote domains and distributes K_1 to those CAs, as depicted by Fig. 2. There are different ways to do it. One way is via the RSA system as follows: (a) Each CA has a public key; (b) the local CA uses the public key of the corresponding CA to encrypt K_1 by the RSA algorithm; (c) Each remote CA can decrypt K_1 using its own private key;
(5) Each remote CA uses its own hierarchy and the DLKM mechanism to distribute K_1 to related resources;
(6) Then, the group is formed and the group members can communicate with one another securely knowing the common key K_1. The corresponding hierarchical monitoring or data access is also allowed.

2.3.3. Data grid

The above schemes are suitable for the computational grid services in which the computing resources are allocated to share the workload. The data communication in this case is mostly generated from the computational task. However, there is another type of grid services which is called the Data Grid whose purpose is mainly data access. Many data sources are offered as resources for other grid nodes/members to access. For example, some bioinformatics problems were solved by the data grid via mining different public databases through the Internet, see e.g. [10].

Nevertheless, some data sources may contain sensitive data and do not allow unauthorized members or machines to access. This condition is very important for further development of data grids. However, the current grid computing schemes make a data source unable to deny some data requests during the grid service because these requests are issued by other nodes that are assigned by the RMS. Moreover, the data source does not know in advance who requests such type of service because the service request also goes to the RMS from users. In this case, many sensitive data sources dare not dedicate themselves into the data grid due to the dearth/loss of access control on their own data. The proposed DLKM can solve this concern.

The above procedure for computational grids can be easily adjusted to the data grid for access control. The steps of distributing the common key K_1 are similar. Moreover, for a certain sensitive data source which needs to confine its data access only to trusted users or machines, the data source should not use the common shared key K_1 to encrypt the data. Rather, it uses its own group private key k_i (suppose it belongs to vertex/group v_i). Then, only those users who are its ancestors and allowed/trusted to read the data can derive k_i via the hierarchical relation (as Fig. 1). The following protocol should be abided by:

(1) Sensitive data sent from a data source should be encrypted by its group private key k_i;
(2) When a trusted machine receives the encrypted data, it can derive k_i;
(3) On the other hand, if a machine is not a trusted one, it cannot get the correct key k_i. In this case, it will report to the RMS and then the RMS will reassign the job to another machine (computational resource) that is trusted by the data source according to the publicized hierarchy of groups.
(4) Then, once a machine obtains k_i, it can decrypt the data, and then operates on the data. If the results out of the sensitive data are also sensitive, then the machine encrypts the results with k_i again before sending back. The encrypted results are sent to the member/user who requested this grid service either directly or through the RMS;
(5) When the user obtains the encrypted sensitive results, he can use his own group private key to derive k_i and then decrypts the results. If he has no right (i.e. neither an ancestor nor in a same group of the sensitive data source), the results are indecipherable to him without deriving the correct key k_i.

By this DLKM mechanism, data sources can also control access to their data using keys and encryption without violating the advantages of the grid computing (large-scale resource sharing). Only those trusted members or sites can derive the key and understand the data, whereas other unauthorized users are unable to understand the encrypted message even though they may be recruited in the grid task and know the common key K_1. Apparently, the hierarchical design that defines the relations of trust/accessibility is very important. If certain data sources are not satisfied with the CA's hierarchical design, they can suggest the CA to modify or select to quit if the CA cannot accept their requirements. Such strategy between the CA and members is bi-directed via our DLKM scheme, which is flexible and fair to both parties who consent on the publicized hierarchy for maintaining appropriate security levels.

2.3.4. Non-monitored grid service

For certain special users (such as a government), they have the right to select a non-monitored option for requesting some special grid services. This means they do not want other people (such as an administrator) to monitor their requested services/results. This requirement is easy to solve by our DLKM along the following steps:

(1) The special request arrives at the RMS and the RMS assigns the available resources to finish the service while notifying the CA with the non-monitored requirement.
(2) The CA randomly generates a key K_1, and then uses the first-level mechanism (rather than the second-level) in our DLKM to hide and distribute K_1 directly. That is, the CA computes $P(x) = \prod_{i\in\psi}(x - f(SID_i, z)) + K_1$ and publicizes $(z, P(x))$ where ψ is the set of involved resources and SID_i are the first-level secrets of those resources.
(3) Then, the involved resources can derive K_1 whereas others, including those ancestors in the hierarchy, cannot obtain K_1.

Thus, the non-monitored grid services can also be offered by our DLKM. Note that not everybody is allowed to request such non-monitored services. Only those special users who need to be strictly authenticated can request them. Without monitoring, some malicious users are easily able to abuse the power of grid computing for their evil purposes.

3. Security and algorithm analysis

This section analyzes the complexity, robustness and other measures of the proposed scheme.

3.1. Security degree

3.1.1. Internal attacks

Internal attack means the users in the same node try to find, individually or by collusion, something they do not know. However, in our scheme, the users in the same node v_i share the same secret key $\hat{k}_i$ and private key k_i. There is nothing except SID_i which one user knows but other users do not know. The only possible attack by an internal individual is that an internal spy attempts to find *SID*s of other users in the same group from $P(x)$. However, our scheme at the first level defends well against this type of attack in an elegant way: the internal spy can obtain K and then subtract K from $P(x)$ to get $A(x)$. By setting $A(x) = 0$, the internal spy tries to find the root. Even though the internal spy may find a root somehow, the root will be $f(SID, z)$ but not *SID*. In addition, one cannot get *SID* from $f(SID, z)$. The only benefit of knowing $f(SID, z)$ is to get K by plugging it into this very $P(x)$ (For any other $P'(x)$, its z' is different, so does $f(SID, z')$). However, the internal spy had got K already. Thus, the clever utilization of one-way function can prevent the internal individual attack.

As for the collusion of multiple internal users in the sense that w internal spies collude to derive other members' personal secrets using all of their SID_i ($i = 1, 2, 3, \ldots, w$), our novel idea of $A(x)$ can make such collusion meaningless. This is very different from other polynomial based techniques [6,5,31]. In those previous polynomial based schemes, $t + 1$ or more internal users may find the entire polynomial by interpolating their $t + 1$ points (i.e. $(ID_i, h(ID_i))$) (where t is the system security parameter and is the degree of the polynomial in general). However, polynomial interpolation is useless here because the users do not have points but just one value (i.e. SID_i but no $A(\mathrm{SID}_i)$) or to say, $f(SID_i, z)$ but $A(f(SID_i, z)) = 0$. Again, $f(SID_{\mathrm{root}}, z)$ may be obtained but SID_{root} cannot. In the next activity when the random value z is changed to z', this last $f(SID_{\mathrm{root}}, z)$ becomes useless. In summary, this DLKM is perfect against internal collusion of any degrees.

3.1.2. Attack across groups

Attacking across groups means the members in different vertices/groups collude to find other group private keys k_i or other users' secret keys SID_i. It may occur in different scenarios, e.g. two users in different child nodes collude to get the key used in their parent node, one user in a sibling node and another user in a child node collude to attack parents.... However, no matter what kinds of combinations they are, their attacks are useless in our DLKM. At the first level, external collusion is meaningless because all the $P(x)$s, no matter whether they are in different or same vertices, are independent due to the random z which is selected every time. The internal attack has been proved useless as above. Similarly at the second level, the attempt to extract another group's private key k_i has to violate the property of one-way function, thus being impossible.

3.1.3. Information glean

A hacker may attempt to glean many publicized $P(x)$s, dreaming to break the system by analyzing the relation among $P(x)$s. As discussed above, this will not work here since all $P(x)$s are independent.

In short, the proposed DLKM is perfect in defending against both internal collusions or external attacks, just like the gold in Fort Knox!

3.2. Complexity analysis

Assume the number of nodes in the hierarchy is n and the maximum possible number of users in a node is m. The one-way hash function is used at both levels. The time complexity of the one-way function is totally determined by the function itself and independent from both n and m. We will ignore its complexity in the following analysis.

3.2.1. The analysis at the first level

At the first level, there are multiple user groups and each is associated with a node in the hierarchy. The CA manages all user groups, but it manages each of them independently. There are three typical operations: initialization, user join and user leave, and three kinds of complexities: storage, computation and communication. The CA needs to store all the information about the hierarchy, nodes, user groups, and users (personal secrets), thus its space complexity is $O(mn)$. However, since we assume the CA is more powerful than users, this space requirement for the CA is not a problem. For every user, the only thing the user needs to store is his personal secret key SID_i. Thus, the storage requirement at the user end is $O(1)$. Note, this is an important feature which makes the scheme applicable in the devices of limited capability such as PDAs. Let us consider three operations one by one. For initializing a node (i.e. the user group of the node), the CA has to calculate $A(x) = (x - f(SID_1, z)) \cdot (x - f(SID_2, z)) \cdots (x - f(SID_m, z))$ and $P(x) = A(x) + k$ and then multicast $P(x)$ (along with z) to all users in the node. The time complexity for computing $A(x)$ is $O(m^2)$. If $A(x)$ has been computed, the time complexity for computing $P(x)$ is just $O(1)$. Otherwise, the time complexity for $P(x)$ is $O(m^2)$. Multicasting $P(x)$ means multicasting its coefficients. Since $P(x)$ has degree m and $m + 1$ coefficients, the message size is $O(m)$. So the communication complexity for initialization is one multicast and $O(m)$ per multicast.

Let's consider the complexities for the new user join operation. The CA just computes a new $P(x)$ and only unicasts it (along with z) to this joining user. So, the time complexity is $O(m^2)$ and the communication complexity is one unicast and $O(m)$ per unicast. Note in the case when a user joins, the user is not allowed to get the previous key for ensuring backward secrecy, then a new secret key $\hat{k}_i$ needs to be selected and multicast to all users including the joining user in the node. In this case, the communication complexity is one multicast and $O(m)$ per multicast. When a user leaves or is revoked, a new group key must be generated and distributed to all the remaining users in the same node. Similarly, the computation complexity is $O(m^2)$ and the communication complexity is one multicast and $O(m)$ per multicast. Finally, let us consider the computation cost for a user to compute the key k from $P(x)$. Computing k is a simple matter to compute $f(SID_i, z)$ and substitute the result for x in $P(x)$. Since the degree of $P(x)$ is m, the computation complexity is $O(m^2)$. One dynamic scenario is that some users may join and some other users may leave the same group at the same time. The elegance of distributing the key via $P(x)$ is that the CA just includes the SIDs of the new joining users and excludes the SIDs of the leaving users in the formation of new $A(x)$. Thus, multiple joins and leaves can be performed in the same efficiency. The Table 1 summarizes complexities at the first level.

3.2.2. The analysis at the second level

One frequent operation for HAC is key derivation, i.e. a node derives the key of its descendant from its own key. This derivation will follow the path from the node to the descendant and use the one-way function iteratively. Obviously, the longer the path, the more complexity the key derivation. The worst case is $O(n)$. To decrease the key derivation complexity, some methods proposed by Atallah et al. [2] are presented to simplify the second-level computation. The time complexity is relieved by adding some shortcuts. After shortcuts were added, during the process of key derivation, one does not need to follow every edge from the beginning node to the destination node any more. Instead, by using shortcuts, we can jump from one special node to another special node, covering many edges in between.

Table 1
Complexities for the first level

	Space	Computation	Communication
Key storage for each user	$O(1)$		
Key computation for each user		$O(m^2)$	
Key initial distribution		$O(m^2)$	$O(1)$ multicast, $O(m)$ per multicast
Key update in joining phrase		$O(m^2)$	$O(1)$ unicast, $O(m)$ per unicast
Key update in leaving phrase		$O(m^2)$	$O(1)$ multicast, $O(m)$ per multicast
Key update for multiple joins and leaves simultaneously		$O(m^2)$	$O(1)$ multicast, $O(m)$ per multicast

Table 2
Time complexities for the second level

Dynamic operations	Without shortcuts	With shortcuts
Key derivation	$O(n)$	$O(\log \log n)$
Add a leaf/internal node	$O(F)/O(F+S)$	$O(n)/O(n)$
Delete a leaf/internal node	$O(1)/O(n)$	$O(1)/O(n)$
Add an edge	$O(1)$	$O(n)$
Delete an edge	$O(1)$ or $O(n)$	$O(n)$

As a result, for an n node hierarchy with shortcuts, the key derivation time complexity is $O(\log \log n)$ one-way function computations with $O(n)$ public space [2].

The typical dynamic operations at the second level include adding/deleting a node/edge. In addition, the shortcut operation may be combined with each of these operations. Let us analyze them one by one; results are shown in Table 2.

When adding a node (it is assumed that the edges between the node and its parents (if any) as well as its children (if any) are also added), the time complexity depends on the position where the new node is added. If the new node is added as a leaf node without any descendants, the time complexity should be constant because only the edge value between the new node and its parent is created. If it is possible/allowed for a node to have multiple parents, the complexity depends on the number of parents. Assume this number is F, then the complexity is $O(F)$. When the new node is added as a parent of some existing nodes, the edge values between the new node and its children also need to be computed. Suppose the number of children is S. The time complexity is $O(S)$. If shortcuts are used, we have to recompute shortcuts. The computation for shortcuts is in linear time of n. Thus the total time complexity is $O(n)$ (Shortcuts are created between centroids and the root. When a new node is added or deleted, the position of centroids will move. We have to compute a shortcut again).

Let us consider the operation of deleting a node. When the removed node is a leaf, nothing needs to be done except discarding the parameters related to the node, so the time complexity is constant. However, if the removed node is an internal node, the situation is quite different. The ID_i used in the descendant nodes will be changed. k_i will also be recomputed through $k_i = f(\hat{k}_i, ID_i)$. All edge values related to these changed ID and k will be recomputed. Since the extreme case is that all the nodes are the descendants of the deleted node, the worst case time complexity for deleting a node is $O(n)$. In addition, if shortcuts are used, shortcuts will be also recomputed which is also in the linear time of n. Thus, the time complexity is $O(n)$.

Let us consider the time complexity for adding/deleting an edge. For adding an edge, the CA just computes the edge value, which is constant. If shortcuts are used, they need to be recomputed, which is $O(n)$. As for deleting an edge, the CA may just discard the edge value. In case shortcuts are used, there is a need to recompute shortcuts, which is $O(n)$. One issue is that if the deleted edge was the only way for the patent node u of the deleted edge to get to its child node s previously, u should not reach s anymore after deletion. This means that all ID values of s and its descendants need to be changed. So all the private keys of these nodes will be recomputed and all the edge values related to these key and ID values need to be recomputed. This complexity is $O(n)$.

3.2.3. *The analysis for combination of two levels*

When a member joins an existing node v_i, what needs to be done is to only distribute the existing $\hat{k}_i$ to this new member with a new $P(x)$. The time complexity will be the same as that for the user join the operation at the first level. It is $O(m^2)$.

When a member is removed from a node v_i, the situation is more complicated. The CA will not only change $\hat{k}_i$, select a new ID_i and recompute k_i but also select a new ID_j and recompute k_j for all its descendant nodes v_j. After that, all edges' values in the sub-tree rooted at v_i will be recomputed. These steps are in linear time of n. Then the CA needs to compute $P_i(x)$ by excluding the removed member and distribute $P_i(x)$ to all the remaining members in v_i. This will take $O(m^2)$. Thus, the total time complexity is $O(m^2+n)$. If shortcuts are used, although no node is deleted and the structure of hierarchy does not change, we also have to recompute the shortcuts because many edge values have been changed. This will contribute another $O(n)$. So, the total time complexity is $O(m^2 + n)$.

From the above description, we can see that when a member moves from one node to another, he was just removed from one node and then joins the other. Thus the time complexity is the sum of complexities for joining and leaving, being $O(m^2 + n)$ too.

In summary, DLKM has very good performance with efficient support for dynamics.

4. Illustrative example

The proposed DKLM has been implemented in a study of the grid computing case in our TEGO (Trusted Electronics and Grid Obfuscation) centre (tego.iupui.edu). To help readers understand better, this case study contains two parts: one is an illustration of a simplified example to show the key

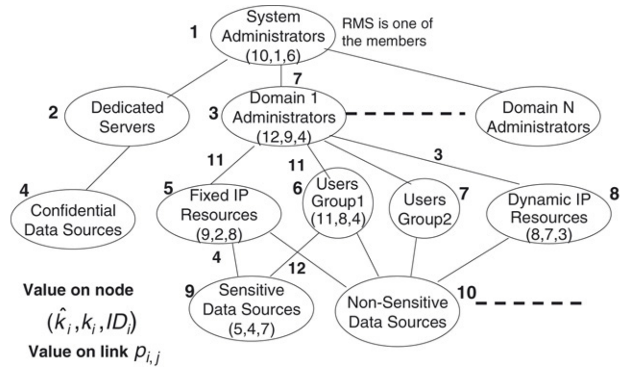

1
System Administrators (10,1,6)
RMS is one of the members
7
2
Dedicated Servers
3
Domain 1 Administrators (12,9,4)
Domain N Administrators
11
11
3
4
Confidential Data Sources
5
Fixed IP Resources (9,2,8)
6
Users Group1 (11,8,4)
Users Group2
7
Dynamic IP Resources (8,7,3)
8
4
12
Value on node
$(\hat{k}_i, k_i, ID_i)$
Value on link $p_{i,j}$
9
Sensitive Data Sources (5,4,7)
Non-Sensitive Data Sources
10

Fig. 3. A part of the sample hierarchy for grid computing.

generation/publication/derivation processes; while the other is the practical implementation in a real case. Suppose the prime is $P = 13$ and the one way function is $f(x, y) = 2^{x \oplus y} \bmod P$.

4.1. Initialization

Suppose the hierarchy at the second-level is designed as Fig. 3. Suppose there are m users in the group represented by the middle vertex 6. For illustration, we use a simplified example with $m = 2$ users. User 1 has personal secret $SID_1 = 7$ and User 2 has $SID_2 = 9$. Then, the group secret key needs to be distributed to both users (suppose $\hat{k}_6 = 11$ and $z = 5$). By Eqs. (1) and (2), the following polynomial is generated:

$$\begin{aligned} P(x)\%13 &= \{A(x) + K\}\%13 \\ &= \{(x - 2^{7\oplus 5})(x - 2^{9\oplus 5}) + 11\}\%13 \\ &= x^2 - 5x + 2. \end{aligned} \tag{8}$$

Then, the coefficients of polynomial (8) are publicized in an array as $\{1, -5, 2\}$. User 1 computes $f(SID_1, z) = 2^{7\oplus 5} = 4$ and substitutes 4 for x into Eq. (8) to obtain $-2\%13 = 11$. User 2 obtains 11 too. Thus, both of them get $\hat{k}_6 = 11$. Suppose another user not in this group with $SID_3 = 6$ to substitute his $f(SID_3, z)$ (=8) into Eq. (8), then he derives $26\%13 = 0$ which is not the key.

The above is just an illustrative example. In our real case study, the system prime P is not so small as 13 but a number with 128 bits. The number m of users in a group should also be much more than two, and the degrees of polynomials are much higher than three. In addition, a strong secure one-way hash function other than the one above can be used.

Generating polynomials is one of the most time-consuming operations. Its performance (running time) for different group sizes from 5 to 3000 is depicted by Fig. 4. Fig. 4(a) shows the efficiency of the algorithm. Usually the number of group members will not be so much up to 3000. Otherwise, the group members can be divided into multiple sub groups. Fig. 4(b) validates that the complexity of this algorithm is $O(m^2)$.

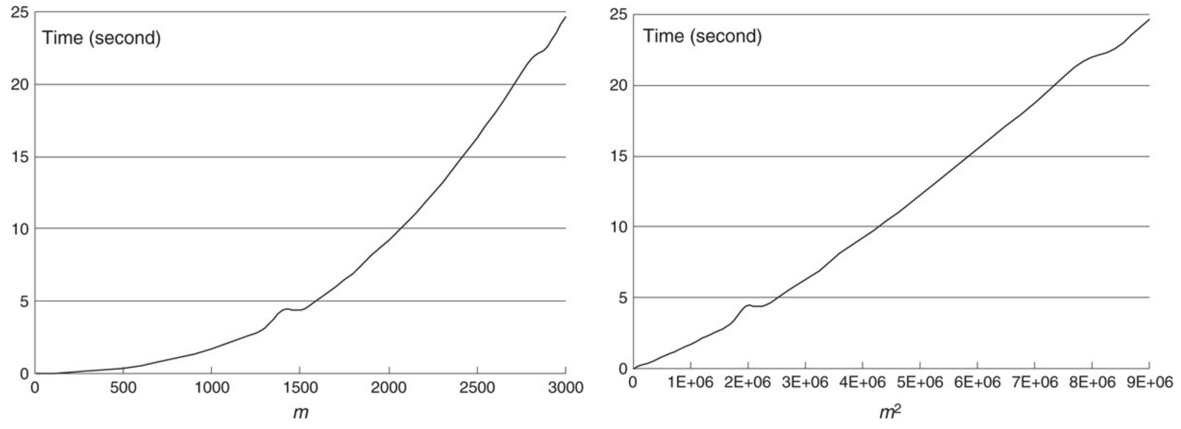
As another example to illustrate the second level's key creation and derivation, we will show how users compute private group keys with the received secret information (i.e. secret group keys), and how users use their private keys and public information to derive the private group key of a descendant vertex. Suppose ID_6 was selected as 4 and $\hat{k}_6$ as 11. Vertex 9 has $ID_9 = 7$ and secret key $\hat{k}_9 = 5$. After the secret keys were distributed to users in two vertices, the private key used in group 6 can be computed by two users as $k_6 = f(\hat{k}_6, ID_6) = 2^{11\oplus 4}\%13 = 8$. Similarly, every user in group 9 can compute private key $k_9 = f(\hat{k}_9, ID_9) = 2^{5\oplus 7}\%13 = 4$. Furthermore, the KMS computes and publicizes the edge value from vertex 6 to vertex 9, that is, $p_{6,9} = k_9 \oplus f(k_6, ID_9) = 4 \oplus 2^{8\oplus 7}\%13 = 12$. When any user in vertex 6 wants to visit resources in vertex 9, he just needs to plug k_6, ID_9, and $p_{6,9}$ in $k_j = p_{i,j} \oplus f(k_i, ID_j)$ to compute $k_9 = p_{6,9} \oplus f(k_6, ID_9) = 12 \oplus 2^{8\oplus 7}\%13 = 4$. Once this group key is obtained, all resources in vertex 9 are available.

4.2. Illustration by a grid service

Now, suppose user 1 in vertex 6 requests a grid service that needs to use sensitive data sources in vertex 9. The available resources assigned by the RMS belong to both vertex 5 and vertex 8. The private and public information on both vertices and links of those involved elements are marked in Fig. 3.

As shown in the part of the data grid in Section 2.3, suppose a common key used by all four groups (and their ancestors) is randomly selected as $K_1 = 3$ by the CA. Along the hierarchy, the CA should encrypt K_1 with the key of group 8 ($k_8 = 7$) once and with the key of group 9 ($k_9 = 4$) another time. Then, the CA multicasts the encrypted key to group 8 with the former one, and to groups 1,3,5,6,9 with the latter one. Groups 8 and 9 can directly decrypt $K_1 = 3$. If a member in group 1 (such as the RMS) needs to get K_1, it can find a path from group 1 to group 9 and then derive the keys of those intermediate vertices one by another along the public information of links and nodes. For instance, the RMS can get $k_3 = p_{1,3} \oplus f(k_1, ID_3) = 7 \oplus 2^{1\oplus 4}\%13 = 9$, then $k_5 = 11 \oplus 2^{9\oplus 8}\%13 = 2$, and finally $k_9 = 4 \oplus 2^{2\oplus 7}\%13 = 4$. Thus, the RMS can decrypt K_1 using k_9 from the multicast message it received. Similar procedures are applicable to other groups. Then, common data can be securely exchanged among them with $K_1 = 3$.

On the other hand, the sensitive data from a data source in group 9 will be communicated with $k_9 = 4$ (not K_1). Thus, the sensitive data can be protected from unauthorized resources/users such as the dynamic IP resources in group 8 though they know the common key $K_1 = 3$. For instance, the computational resources in group 5 access the sensitive data from a database in group 9. The sensitive data is encrypted by $k_9 = 4$ that can be derived by group 5. Then, these computational resources compute the results from the decrypted data, encrypt the results with k_9, and send the results to the RMS in group 1. After composing all results, the RMS returns the final results to the user in group 6 who requested for this service. The insensitive results in the final results are encrypted by $K_1 = 3$ whereas sensitive results are encrypted by $k_9 = 4$. Thus, the authorized users can derive both keys and know the entire results. In case some malicious resources in group 8, holding $K_1 = 3$, attempt to access the sensitive data, or some resources are accidentally assigned by the RMS for operating on the sensitive data, those malicious attempts



(a) Running time vs. group members (m). (b) Time is linear to m^2.

Fig. 4. Performance for generating polynomials to numbers of group members.

from unauthorized users cannot understand the data without the correct key ($k_9 = 4$). Moreover the accidentally assigned resources can report to the RMS to reassign other authorized resources to continue the jobs. As a result, a data source can also control its data privacy through the above process rather than makes itself known by all involved or assigned resources/users.

5. Discussions

Cryptographic key and key management have been intensively investigated for secure group communication (SGC), e.g. [7,12,22] and hierarchical access control (HAC), e.g. [8,28]. The group communication is further considered in two formats: one-to-many multicast e.g. [29], and many-to-many communication, e.g. [13]. In terms of group key management for SGC, typical existing group key management schemes can be classified as four categories [30]: (1) centralized key distribution, e.g. key graph and one-way function chain (OFC) [9]; (2) decentralized key management, e.g. Iolus [23]; (3) distributed/contributory group key agreement, e.g. DISEC [13]; and (4) distributed group key distribution, e.g. optimized group key rekeying [26].

For HAC key management, Akl and Taylor [1] proposed the first cryptographic HAC scheme and then many other HAC schemes followed. Typical existing cryptographic HAC mechanisms can be classified as [30]: unconditionally secure and conditional secure. The former assumes node keys are totally independent whereas the latter establishes dependent relations among node keys using a one-way function. The latter is further classified as directly dependent key and indirectly dependent key. Direct dependence means that a child's key is directly computed from its parent key. Only the roots' keys are randomly selected and all other keys are computed. The first cryptographic HAC scheme [1] belongs to this class. In contrast, indirect dependence means that all nodes' keys are randomly selected and they are independent, but there are some public information computed, via a one-way hash function, from the keys of two nodes which have parent–child relation. From the public information, a parent node can compute the key of its child.

As it is well recognized, grid computing intensively involves collaborative tasks which multiple entities work together to complete via interactions, data exchange and resource sharing. Thus, secure group communication and hierarchical access control would be necessary in the grid security architecture. Our proposed Dual Level Key Management solution is the first to integrate SGC and HAC systematically for secure grid computing services. In addition, the prior schemes pose certain assumptions which are difficult to implement in reality and/or have some problems. For example, most schemes require that group members be organized/ordered as a tree (called as member serialization) which exposes group members and their positions. They also require multiple encryptions and/or multiple round broadcasts for key updating. In contrast, our ACP based key distribution solution has no serialization issue, can hide membership, and is able to perform rekeying with just one polynomial multicast, no matter how many members join or leave.

6. Conclusion

In this paper, we proposed a Dual Level Key Management scheme and analyzed its security and performance. The scheme is highly secure and efficient and is able to support highly dynamic operations at both user level and group level. We also showed how the scheme can be used for securing different Grid-based services. The illustrative example showed the efficiency of the proposed DLKM and validated its correctness. It also helped readers to clearly understand the proposed DLKM and know how to implement it in real grid computing services.

Acknowledgments

The authors sincerely appreciate the constructive comments from journal editors and anonymous reviewers. This work was partially supported by US NSF grant CCR-0311577.

References

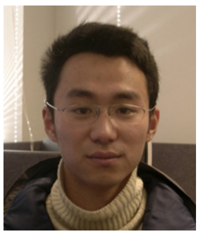
[1] S.G. Akl, P.D. Taylor, A cryptographic solution to the problem of access control in a hierarchy, ACM Transactions on Computer Systems (TOCS) 1 (3) (1983) 239–248.

[2] M.J. Atallah, K.B. Frikken, M. Blanton, Dynamic and efficient key management for access hierarchies, in: ACM Conference on Computer and Communication Security, CCS 05, 2005, pp. 190–202.
[3] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, D. Zagorodnov, Adaptive computing on the Grid using AppLeS, IEEE Transactions on Parallel and Distributed Systems 14 (14) (2003) 369–382.
[4] M. Bishop, Computer Security: Art and Science, second ed., Addison Wesley, ISBN: 0-201-44099-7, 2003.
[5] C. Blundo, L.A.F. Mattos, D.R. Stinson, Generalised beimel-chor scheme for broadcast encryption and interactive key distribution, Theoretical Computer Science 200 (1998) 313–334.
[6] C. Blundo, A.D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, M. Yung, Perfect secure key distribution for dynamic conferences, in: Advances in Cryptology, CRYPTO'92, in: LNCS, vol. 740, Springer, Berlin, 1993, pp. 471–486.
[7] A. Boukerche, C. Dzermajko, K. Lu, Alternative approaches to multicast group management in large-scale distributed interactive simulation systems, Future Generation Computer Systems 22 (7) (2006) 755–763.
[8] G. Bu, Z. Xu, Access control in semantic grid, Future Generation Computer Systems 20 (1) (2004) 113–122.
[9] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, Multicast security: A taxonomy and some efficient constructions, in: Proceedings of the IEEE INFOCOM. vol. 2, 1999, pp. 708–716.
[10] Y.S. Dai, M. Palakal, S. Hartanto, X. Wang, Y. Guo, A grid-based pseudo-cache solution for MISD biomedical problems with high confidentiality and efficiency, International Journal of Bioinformatics Research and Applications 2 (3) (2006) 259–281.
[11] S.K. Das, D.J. Harvey, R. Biswas, Parallel processing of adaptive meshes with load balancing, IEEE Transactions on Parallel and Distributed Systems 12 (12) (2001) 1269–1280.
[12] E. Dawson, A. Clark, M. Looi, Key management in a non-trusted distributed environment, Future Generation Computer Systems 16 (4) (2000) 319–329.
[13] L. Dondeti, S. Mukherjee, A. Samal, DISEC: A distributed group key management scheme for secure many-to-many communication, in: Fifth IEEE Symposium on Computers and Communications, Antibes-Juan les Pins, France, 2000, pp. 693–698.
[14] I. Foster, C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 2003.
[15] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke, Grid services for distributed system integration, Computer 35 (6) (2002) 37–46.
[16] I. Foster, C. Kessekan, G. Tsudik, S. Tueckel, A security architecture for computational grid, in: Proceedings of ACM Conference on Computer and Communication Security, CCS, 1998, pp. 83–92.
[17] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, International Journal of High Performance Computing Applications 15 (2001) 200–222.
[18] C. Kaufman, R. Perlman, M. Speciner, Network Security: Private Communication in a Public World, second ed., Prentice Hall, ISBN: 0-13-046019-2, 2002.
[19] A. Kumar, An efficient SuperGrid protocol for high availability and load balancing, IEEE Transactions on Computers 49 (10) (2000) 1126–1133.
[20] J. Li, D. Cordes, A scalable authorization approach for the Globus grid system, Future Generation Computer Systems 21 (2) (2005) 291–301.
[21] C.H. Lin, Dynamic key management scheme for access control in a hierarchy, Computer Communications 20 (15) (1997) 1381–1385.
[22] I.-C. Lin, M.-S. Hwang, C.-C. Chang, A new key assignment scheme for enforcing complicated access control policies in hierarchy, Future Generation Computer Systems 19 (4) (2003) 457–462.
[23] S. Mittra, Iolus: A framework for scalable secure multicasting, Journal of Computer Communication Reviews 27 (4) (1997) 277–288.
[24] C.P. Pfleeger, S.L. Pfleeger, Security in Computing, third ed., Prentice Hall, ISBN: 0-13-035548-8, 2003.
[25] D.J. Power, E.A. Politou, M.A. Slaymaker, A.C. Simpson, Securing web services for deployment in health grids, Future Generation Computer Systems (5) (2006) 547–570.
[26] O. Rodeh, K. Birman, D. Dolev, Optimized group re-key for group communication systems, in: Network and Distributed System Security, CA, USA, 2000.
[27] D.R. Stinson, Cryptography: Theory and Practice, second ed., CRC Press, ISBN: 1-58488-206-9, 2002.
[28] L. Wang, K. Chen, Comments on a theorem on grid access control, Future Generation Computer Systems 22 (4) (2006) 381–384.
[29] N. Weiler, SEMSOMM: A scalable multiple encryption scheme for one-to-many multicast, in: Proceedings of the 10th IEEE International WETICE Enterprises Security Workshop, CA, USA, 2001, pp. 231–236.
[30] X. Zou, B. Ramamurthy, S. Magliveras, Secure Group Communication over Data Networks, Springer, ISBN: 0-387-22970-1, 2004.
[31] X. Zou, S. Magliveras, B. Ramamurthy, A dynamic conference scheme extension with efficient burst operation, Congressus Numerantium 158 (2002) 83–92.

Dr. Xukai Zou is a faculty member with the Computer Science Department of Purdue University School of Science at IUPUI. He received his Ph.D. in Computer Science from University of Nebraska-Lincoln. His research is in Applied Cryptography, Communication Networks and Security, Design and Analysis of Algorithms, and Grid Computing. He has published three books and over twenty articles in these areas recently. Dr. Zou is a recipient of the U.S. NSF Cyber Trust Awards and the leading author of the Books "Secure Group Communication over Data Networks" (Springer) and "Trust and Security in Collaborative Computing" (World Scientific). Dr. Zou is a Program Chair and a Program Committee Member for a number of international conferences and serves on the Editorial Board and as a reviewer for many international journals and conferences.

Dr. Yuan-Shun Dai is Assistant Professor of the Computer Science Department of Purdue University School of Science, at IUPUI. He received his Ph.D. from National University of Singapore, majored in Systems Engineering. He has published three Books, and more than 40 Articles including six IEEE/IIE/ACM Transactions papers. He was featured by Industrial Engineer Magazine (December, 2004 page 51), the most important magazine in industries, due to his research achievement and contribution to the Industries and Engineers. His research is in grid computing, dependability, autonomic computing, security, fault tolerance.

He is a Programme Chair for a regular and prestigious IEEE conference, the 12th Pacific Rim Symposium on Dependable Computing (PRDC2006). He is also a General Chair for the 2nd IEEE Symposium on Dependable Autonomic and Secure Computing (DASC06), a General Chair for IEEE International Workshop on Trusted and Autonomic Computing Systems (TACS06), a General Chair of the 1st IFIP workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES05), and a Programme Chair of the first IEEE workshop on Reliability and Autonomic Management in Parallel and Distributed Systems (RAMPDS05). He serves as the Guest Editor for the Journal of "Computer Science" on a special issue of "Reliability and Autonomic Management", and also serves for many technical/programme committees at international workshops, symposiums and conferences.



Xiang Ran is a graduate student and research assistant in Computer Science Department of Purdue University School of Science at IUPUI. He received his bachelor degree in Information Engineering from Chengdu University of Technology. His research focus is Network Security and Distributed Computing. He is obsessed with system design and computer network technology.