This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2008 proceedings.

A Practical and Flexible Key Management Mechanism For Trusted Collaborative Computing

Xukai Zou
Department of Computer Science
Indiana University Purdue University Indianapolis
Indianapolis, IN 46202, USA
xkzou@cs.iupui.edu

Yuan-Shun Dai
Department of Electrical Engineering & Computer Science
University of Tennessee, Knoxville
Knoxville, TN, 37996-0700, USA
YDai1@eecs.utk.edu

Elisa Bertino
Department of Computer Science
Purdue University and CERIAS
West Lafayette, IN 47907, USA
bertino@cs.purdue.edu

[1]*Abstract*—**Trusted Collaborative Computing (TCC) is a new research and application paradigm. Two important challenges in such a context are represented by secure information transmission among the collaborating parties and selective differentiated access to data among members of collaborating groups. Addressing such challenges requires, among other things, developing techniques for Secure Group Communication (SGC), Secure Dynamic Conferencing (SDC), Differential Access Control (DIF-AC), and Hierarchical Access Control (HAC). Cryptography and key management have been intensively investigated and widely applied in order to secure information. However, there is a lack of key management mechanisms which are general and flexible enough to address all requirements arising from information transmission and data access. This paper proposes the first holistic group key management scheme which can directly support all these functions yet retain efficiency. The proposed scheme is based on the innovative concept of Access Control Polynomial (ACP) that can efficiently and effectively support full dynamics, flexible access control with fine-tuned granularity, and anonymity. The new scheme is immune from various attacks from both external and internal malicious parties.**

Keywords-Information security, Trusted Collaborative Computing (TCC), Key management, Cryptography, Secure Group Communication, Access control.

I. INTRODUCTION

Information and communication technologies along with society's drive for collaboration in the modern world make "*collaborative computing*" (CC) and its applications possible and necessary. Typical CC applications include, but not limited to, multi-party military actions, tele-conferencing, tele-medicine, interactive and collaborative decision making, grid-computing, information distribution, and pay per view services. Trust in such environment can eventually determine its success and popularity due to people's desire for confidentiality, privacy and integrity of their personal and/or corporate information. The current Internet by design does not provide high assurance security for data transmission [1, 2]. Compared to the two-party interaction model (such as the client-server service model), CC environments are group-oriented, involve a large number of entities and shared resources, are complex, dynamic, distributed, and heterogeneous and may possibly include hostile elements. Systems experience failures due to intrusions and attacks from hostile entities [3, 4]. In addition, there is the problem of insider threats, by which attacks are from malicious parties inside the organizations or members of CC groups. Consequently, building a trusted collaborative computing (TCC) environment is very difficult and requires a long term persevering endeavor.

TCC environments are characterized by collaborative tasks which require multiple entities to work together and share their resources. The first key issue in this environment is that multiple participating entities must communicate securely among one another via secure communication channels. IP multicast provides efficient transmission of messages to a group of users; however, the open nature of IP multicast makes it unable to provide strong confidentiality. Thus, secure group-oriented communication is the first fundamental function for TCC. Another key requirement is related to resource sharing and data exchange. Access to shared resources/data must be finely controlled; otherwise attackers and malicious users can access resources to which they are not entitled to access, abuse, tamper, and even damage the resources. Thus selective data sharing, at different granularity levels, along with access control becomes another fundamental function. These two classes of fundamental functions should be sufficiently flexible in supporting various possible forms of interactive access relations between the parties and the resources in the system. We can thus identify four security requirements that are relevant for TCC: hierarchical access control (HAC), secure group communication (SGC), secure dynamic conferencing (SDC), and differential access control (DIF-AC). Cryptography is a powerful tool to support all these four functions. As well known, key management is the most important yet difficult issue in such context. How to generate, distribute, update, and revoke keys in large and dynamic environments is an important challenge.

[1] This research was supported in part by NSF CRI grant 0551410, NSF CT grant 0311577 and IUPUI RSFG grant.

Group key management (GKM) has been investigated in the specific context of all these function classes. In the SGC field, different group key management protocols have been proposed including centralized group key distribution [5, 6, 7, 8, 9, 10, 11], decentralized group key management with relaying [12], (distributed) contributory group key agreement [13, 14, 15, 16, 17, 18, 19], and distributed group key distribution [20]. With respect to conference key management (CKM) for SDC, there are the naive solution [21], the secure lock SDC scheme [22], the Key Tree & interval based SDC scheme [23], and the polynomial based scheme [2, 24]. The cryptographic solution for DIF-AC is also called broadcast encryption and the key management schemes for DIF-AC include the ones in [25, 26, 27]. With respect to hierarchical key management for HAC, the first scheme was proposed by Akl and Taylor in 1983 [28, 29], followed by many others [30, 31, 32].

The existing GKM schemes are good at supporting individual functions. However, there is a lack of a group key management scheme which is general and flexible enough to address various security requirements by TCC applications with a single and holistic manner. An existing GKM scheme for one class of functions cannot be directly and easily used for another. On the other hand, all these functions usually co-exist in TCC applications, are somehow related, and need to be efficiently supported. Combining multiple different GKM protocols from different classes to support all functions is possible but inefficient because it would require multiple sets of modules, storage systems, configurations, coordination, coherence, and conversion among these protocols. In this paper, we propose a generic GKM scheme for TCC as the first holistic GKM scheme that can uniformly support all the above functions, and moreover possesses good performance and flexibility. The proposed scheme is based on the innovative concept, or to say, construction, of an *Access Control Polynomial* (ACP) over the finite field. Such construction is very efficient in supporting highly dynamic environments (e.g., users join/leave, addition/deletion of resources/data/messages, addition and removal of user/resource relations), random user/data structures/formats according to fine-tuned granularity (e.g., in the levels of users, user groups, data sets, data records, record fields), and anonymity (i.e. group membership and size can be hidden from outsiders and insiders). Due to its uniform support for different security functions, the new proposed scheme can easily implement the integration of various application systems. Most importantly, the new scheme is immune from various attacks, including external hackers and internal malicious members, and even their collusion.

The rest of the paper is organized as follows. Section II presents the Access Control Polynomial (ACP) based key management scheme, its application to four fundamental security functions, and analysis of its security and performance. Section III discusses its support for anonymity and other access models and the improved ACP mechanism to reduce its complexity from $O(n)$ to $O(\log(n))$. Section IV presents some concluding remarks.

Abbreviations:

ACP:	Access Control Polynomial
CC:	Collaborative Computing
CKM:	Conference Key Management
DIF-AC:	Differential Access Control
GKM:	Group Key Management
HAC:	Hierarchical Access Control
SDC:	Security Dynamic Conferencing
SGC:	Secure Group Communication
TCC:	Trusted Collaborative Computing

Notations:

$A(x)$:	The Access control polynomial in the form of $A(x)=\prod_{i\in\psi}(x-f(SID_i,z))$
F_q:	The finite field
f :	A public cryptographic hash function. It is used in the form of $f(x,y)$, e.g. $f(x\,\|\,y)$
CID_i :	Secret Class/Group Identifier, a positive integer
$P(x)$:	The public polynomial sent to users for key distribution, $P(x) = A(x)+K$
q:	A large prime, as a predefined system parameter
SID_i:	Personal Permanent Portable Secret, a positive integer
U_i :	A group member in a certain group
c_j :	A certain vertex (i.e. class) in the hierarchy
z :	A random integer which is changed and made public every time.

II. An Innovative Key Management Scheme

First, we introduce our innovative construction of an Access Control Polynomial (ACP) through which secret information can be distributed so that only the intended recipients (i.e. their IDs are included as a term $(x-f(ID,z))$ in the polynomial) can derive that secret information. Then we show its application to four fundamental functions we outlined previously. Finally, the ACP security and complexity are analyzed.

The following assumptions are made in the paper:

1) q is a large prime from which a finite field F_q is formed. 2) $f:\{0,1\}^* \rightarrow \{0,1\}^q$ is a cryptographic hash function. 3) There is a trusted central server. Every valid user, say U_i, in the system is assigned a Personal Permanent Portable Secret, called P3-Secret and denoted as SID_i (a random positive integer $<q$). This secret is only known to the user and the central server. Since users are generally required to register to the system, the assignment of an *SID* to a user can be performed during the registration procedure.

A. *Access Control Polynomial and Secret Information Distribution*

An *access control polynomial* (ACP) is a polynomial over F_q [x] and defined as follows.

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

where ψ denotes the user group under consideration and SID_i are the P3-Secrets of group members in ψ. z is a random integer from F_p and is made public. In addition, z is changed every time $A(x)$ is computed. It is evident 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.

In order to broadcast a secret value such as K to the users in ψ, the following polynomial is computed (by the trusted server): $$P(x) = A(x) + K \qquad (2)$$

Then, $(z, P(x))$ is publicized (broadcast) and K is hidden, mixed with the constant of $A(x)$. From $(z, P(x))$, any group member U_i with SID_i can obtain K by:

$$K = P(f(SID_i, z)) \qquad (3)$$

B. *Application to Various Security Functions*

Based on ACP, the key management problem for a large range of security functions and applications can be solved effectively. This subsection describes how it can be used to provide an effective and efficient key management mechanism for SGC, SDC, DIF-AC and HAC.

Group key management for Secure Group Communication (SGC)

SGC refers to a setting in which a group of members can communicate (or share the information) among themselves, in a way that outsiders are unable to understand the communication (or the information) even when they are able to intercept the communication (or steal the information). The confidentiality of the SGC communication is provided by encrypting the communication with a group key which is distributed to the group (and only group) members.

The server computes $A(x)$ by Eq. (1), $P(x)$ by Eq. (2), and multicasts $(z,P(x))$. Then every user in the group can compute the key via Eq. (3). After all group members obtain the same key, they can conduct group communication securely.

Let us consider dynamics. Users can join, leave or be revoked from the system. From the construction of $A(x)$, it can be seen that regardless of whether we deal with single join, single leave, multiple joins, multiple leaves, or multiple joins and leaves simultaneously, dynamics can be implemented with great elegance: the above steps (1), (2), and (3) are executed but in the formation of A(x), just the joining users' *SID*s (in fact, $f(SID_i, z)$) are included and the leaving users' *SID*s are excluded. Note that z and K in these steps are new random numbers. Once the key is changed, the encryption with the new key will prevent the leaving (or joining) users from accessing the future (or the past) information.

Conference key management for Secure Dynamic Conferencing (SDC)

SDC refers to a scenario where any random subset of the given user population can form a secure communication (sub)group. As evident, SDC is closely related to SGC: as an extension of SGC or equivalently, SGC as a specific case of it. Suppose the size of the universe under consideration is n, there will be 2^n-n-1 possible conferences. Pre-generating all these 2^n-n-1 conferences is a bad strategy because many conferences may never need to be activated. In addition, most conferences may not occur at the same time. However, the existing schemes try to generate all the keys (or information from which to derive all the keys) for all the conferences at the very beginning, thus resulting in exponential running time and/or storage cost.

One specific feature of the ACP scheme is that it is an "on-the-fly" mechanism, which means that whenever there is a need to distribute a secret to a specific user group, just the above steps (1) (2) and (3) have to be executed. This feature is particularly useful for supporting SDC. Whenever there will be a conference of any subset of users, the server just performs the three steps where $A(x)$ includes *SID*s of the conference members. If a user participates in multiple conferences at the same time, the user's *SID* can be included in multiple corresponding $A(x)$s and the user then can get the keys for all these conferences. Whenever users want to join or leave a conference, the above three steps are executed with $A(x)$ just including the intended users. Thus, the dynamics can be efficiently processed in SDC.

Resource key management for Differential Access Control (DIF-AC)

Access control is used for checking whether a user has the right to access a certain resource and for granting or denying access as required. It is a fundamental security issue for many computing systems in which users and resources are involved.

There are three typical policy models for specifying access control policies [33, 34]: Discretionary access control, Mandatory access control and Role-based access control (RBAC). Traditionally, access control has been implemented using non-cryptographic mechanisms such as access control lists, capability lists, and access control matrix. Recently, a new class of access control mechanisms, called cryptographic access control, has been proposed [28]. The idea is that every resource is assigned a cryptographic key and the resource is encrypted with the key. This key is distributed to the users who are supposed to access the resource (e.g. users that have already paid for the resource). Only if the key provided by a user matches the resource key, the resource grants the user's access, otherwise, denies the access. There are two typical access control models in TCC: differential access control (DIF-AC) and hierarchical access control (HAC).

In DIF-AC, a user can (and only can) access certain resources and a resource can (and only can) be accessed by certain users (i.e. many-to-many relation, determined by, for example, subscription and payment). The typical applications requiring DIF-AC include, but are not limited to, e-newspapers, pay-per-view broadcast TV, multiple streaming services.

Like the above SDC scheme, every resource R_k is associated with a dynamic key K_k and the users who can access R_k are treated as a conference. The server computes $A_k(x)$ and $P_k(x)$, and publicizes $(z, P_k(x))$. Thus, the user, who can access R_k, can derive key K_k and is granted access to resource R_k. If a user can access multiple resources, the user's SID_i will be included in the $A_k(x)$s of all these resources. Thus, the user can access all these resources. Similarly, dynamics can be implemented by inclusion and exclusion of users' SIDs in the formation of new $A_k(x)$s.

Hierarchical key management for Hierarchical Access Control (HAC).

HAC occurs when resources (and users) have some hierarchical relation: resources are assigned levels and a user who has the access right to a resource at one level is automatically granted access to the resources which are the resource's children or descendants at lower levels. However the reverse is not allowed. The most generic format of HAC can be represented as a Directed Acyclic Group (DAG) (Figure 1). A node in the hierarchy can represent a user, a resource, a set of users, a set of resources, or both users and resources.

For every node/class C_k in the hierarchy, the server selects a unique CID_k and securely distributes CID_k to C_k's users $\{U_1, U_2, \cdots, U_n\}$ using the same scheme as that in SGC, i.e. the server computes

$$P(x) = (x - f(SID_1, z)) \cdot (x - f(SID_2, z)) \cdots (x - f(SID_n, z)) + CID_k \quad (4)$$

and multicasts $(z, P(x))$ to C_k's users. The server also selects a dynamic key K_k for every C_k. Now, the server constructs $A_k(x)$ using this node's CID_k as well as CIDs of all its ancestors:

$$A_k(x) = (x - f(CID_k, z)) \prod_{i \in \psi} (x - f(CID_i, z)) \quad (5)$$

where the first term is C_k itself and the next terms are associated with all the ancestors C_i of C_k (ψ is the set of ancestors of C_k). Then, the server constructs $P_k(x) = A_k(x) + K_k$ and publicizes $(z, P_k(x))$. The node C_k (i.e., the users in C_k) can compute the key K_k as $K_k = P_k(f(CID_k, z))$. Furthermore, any ancestor (i.e. the users in) C_i of C_k can also derive the key K_k as $K_k = P_k(f(CID_i, z))$. However, C_k cannot reversely get C_i's key. Thus, the hierarchical access control is correctly and securely enforced.

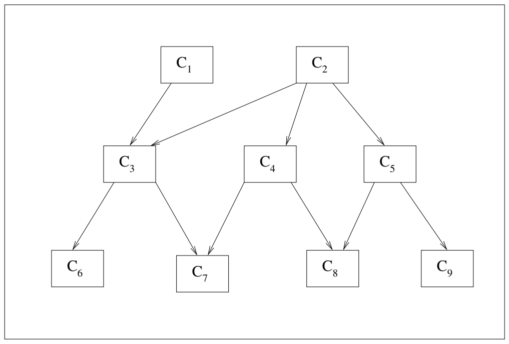

Figure 1: A typical access control hierarchy.

In this ACP based HAC scheme, the key derivation by the node's ancestors is performed in the identical way as the key computation by a node. Moreover, nodes do not need to know the exact hierarchy. The nodes that are ancestors of a node will obtain the correct key of the node when substituting their CID into $P(x)$ but others will not. This is an important advantage of our approach compared with other cryptographic HAC approaches.

There are two level dynamics in HAC: node level and user level. Node level dynamics includes adding a node, deleting a node, moving a node from one place to another, adding one link between two nodes, and deleting a link between two nodes. User level dynamics indicates addition and deletion of a user from a node group and movement of a user from one node group to another. Based on ACP, both level dynamics can be accomplished efficiently.

Let us consider the operation of deleting a node, since revocation/deletion is generally more difficult to deal with than join/addition. There are two cases to consider: a leaf node and an internal node. If the deleted node is a leaf node, nothing needs to be done except just discarding the information/values related to this node. If the deleted node is an internal node, a policy needs to be adopted in order to relocate the node's children. However, the policies used for such purpose do not matter here. Since the deleted node knew the keys of all its descendants, these keys need to be changed, which is easy. For each of the descendant nodes of the deleted node, the server computes $A(x)$ which includes the *CID*s of all new ancestors of the node but excludes the *CID* of the deleted node and multicasts (z, $P(x)=A(x)+K$).

Consider the second level dynamics. For example, if one member (with SID_l) leaves group C_k and attends another group C_j, the following two steps complete the update.

(1) The new node *CID* of C_k is updated by the above polynomial excluding the term $(x - f(SID_l, z'))$.

(2) The new node *CID* of C_j is updated with the above polynomial including the term $(x - f(SID_l, z''))$. (Note: new z' and z'' are used).

It is clear that the ACP mechanism can address the HAC problem in the same manner and the same efficiency of SGC/SDC. Typical applications involving HAC include digital libraries and medical information systems.

C. *Security and Complexity Analysis*

We now analyze the security and performance of the above ACP scheme. By the security analysis, we show that the proposed ACP mechanism is very robust and secure not only against outside attackers which do not know the shared key but also against the insiders which know the shared key. By the performance analysis, we show that the ACP mechanism is efficient. In particular, the improvement based on hierarchical grouping (See Section III.C) can make the ACP mechanism reach $O(\log(n))$ complexity.

Security of the ACP scheme

We discuss the security of the scheme in terms of external attackers, internal attackers, and collusion of attackers. First, let us consider the key space and the guessing or brute-force attack. K is randomly and uniformly selected from 0 to q-1. In addition, K can be coincident with any of SID_i and $v_i = f(SID_i, z)$, for $i = 1, \cdots, n$ since this coincidence will not affect the correctness of the ACP mechanism. Thus, the introduction of the access control polynomial (no matter how high its degree is) will not reduce the size of the key space. As for the brute-force attack, an external attacker can either guess K directly or guess one of v_i and then compute K, or guess one of SID_i and compute v_i and then K. The probability that a random guess hits K is $1/q$ whereas it is n/q to hit any of v_i and another n/q to hit any of SID_i. Thus, the overall probability for a random trial to succeed is $(2n+1)/q$. This means that the access control polynomial increases the success chance of the brute-force attack by a factor of $2n$. The more users are included in the polynomial, the higher the probability of success by the brute-force attack. However, due to the efficiency of the ACP mechanism (as discussed below), q can be selected to be reasonably large (e.g. 128 bits), thus, making the brute-force attack infeasible. Next, let us consider the attacks in which an external attacker tries to obtain the group key K or group users' *SID*s from $P(x)$. The K is hidden in the publicized constant term of $P(x)$, i.e. $c_0 = (K+V)\%q$ where $V = v_1 \cdot v_2 \cdots v_n$ and $v_i = f(SID_i, z)$, for $i = 1, \cdots, n$. Since there are many other pairs of K' and V' such that $c_0 = K' + V'$, the attacker cannot uniquely determine K from c_0. As for trying to determine all of $K, v_1, v_2, \cdots, v_n$ from (the coefficients of) $P(x)$ at the same time, the attacker will fail because only n equations can be formed for n+1 unknown $K, v_1, v_2, \cdots, v_n$. As for trying to determine SID_i, the only relevant value is $v_i = f(SID_i, z)$ which is difficult to be obtained from $P(x)$ as discussed above. Even if the attacker was able to determine $v_i = f(SID_i, z)$ somehow, the attacker still would not be able to get SID_i since this would require inversion of the cryptographic hash function f. Finally, multiple external attackers may collude to determine K or SID_i, but their collusion provides no more information than the information that would be obtained by a single attacker; collusion is thus useless. As a result, ACP is resistant to external attacks.

We now consider the case of internal malicious users. Obviously, an internal user can obtain K from its own SID_i. Thus the purpose of an internal malicious user is to obtain the *SID*s of some other users so that he can get the secret information, reserved to other users, to which he is not authorized to access. He can obtain the exact polynomial $A(x)$ as $A(x)=P(x)-K$ and then set $A(x)=0$ to determine the roots of $A(x)$. He may find $v_i = f(SID_i, z)$ by some root-finding algorithm. However, it is computationally infeasible to get SID_i from $v_i = f(SID_i, z)$ due to the one-way feature of *f*. Getting v_i of another user does not help the attacker. First, v_i will result in *K* to be disclosed, but this does not help at all because he had been allowed to get *K* from his own *SID*. Additionally, this $v_i = f(SID_i, z)$ can be only used for getting this *K* and cannot help in determining any other keys from other *P*(*x*)s because *z* is updated every time and two v_is in two *P*(*x*)s will be different even though SID_i is the same. As a result, the internal malicious user cannot compromise the scheme. Furthermore, it is useless for multiple internal users to collude because their collusion cannot help to make the inverse of the cryptographic hash function easier, thus, making it impossible to get SID_i from v_i. The collusion of internal malicious users and external attackers is also useless in getting other users' *SID*s (Note: the collusion here does not include the case of an internal user giving his *SID* or the key to an outsider so that the outsider can access the information. If this case is considered as collusion, then it is inherent in all cryptosystems and there is no solution to it).

The attackers may hope to glean multiple *P*(*x*)s and try to get useful information from them; however, this attempt would also be useless due to the changing *P*(*x*)s. There are different forms of collusions in the hierarchy such as two siblings trying to figure out their parent's key, a node and its nephew trying to figure out its parent key. However, these cases of attacks can be reduced to the collusion of external attackers, or internal malicious members or internal/external users depending on whether (and how many) their *SID*s are included in *P*(*x*). As discussed above, our ACP scheme is able to perfectly defend against any such collusion.

We note that the system is secure against repeated join and leaving attacks. However, if a previous *z* is re-used later, a user U_i belonging to the current group is able to obtain another user U_j's $v_j = f(SID_j, z)$ and thus get the key of the previous group which had the same value *z* and to which U_j belonged to. So the trusted server should check and make sure that a fresh random *z* is used every time.

Complexity of the ACP scheme

The storage complexity (at both user end and server end), computation complexity (at both the user end and server end), and communication complexity are analyzed here.

The user-end storage cost is *O*(1) since a user just needs to store its P3-Secret *SID* (plus its node *CID* if in the HAC hierarchy). The server storage cost is *O*(*n*+*m*) since the server needs to store all *n* users' *SID*s (plus *m* nodes *ID*s if in the HAC hierarchy). Suppose there are *n* terms involved in the generation of *P*(*x*). There are two parts to consider. The first part is related to computing $f(SID, z)$. The running time of the cryptographic hash function totally depends on itself but is independent from the number of terms *n*. Suppose its running time is *O*(*B*), then computing *n* $f(SID, z)$s has a cost in *O*(*nB*). The other part is to multiply *n* terms (*x*-*v*)s. The main operations are multiplication (with modulo) and addition (with modulo). There are in total $O(n^2)$ such operations. The computation complexity for multiplying *n* terms (*x*-*v*)s is in $O(n^2)$. Thus, the total computation complexity for generating *P*(*x*) is in $O(nB + n^2) = O(n^2)$. This polynomial computation complexity is efficient for the server. We now consider the computation complexity for computing the key from a polynomial *P*(*x*) of degree *n* when replacing *x* with the computed value $v = f(SID, z)$. The main operations are:

(1) the computation of $v, v^2\%q, \cdots, v^n\%q$ which requires *n* multiplications (with modulo); (2) the multiplication of each of these values with its corresponding coefficient, which requires another *n* multiplications; and (3) the addition of the results, which requires *n* additions. In total, the complexity of computing the key from *P*(*x*) is in $O(n)$. With respect to the communication complexity, broadcasting $P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ requires to broadcast the coefficients $a_n, a_{n-1}, \cdots, a_1, a_0$. Thus, the communication complexity is in *O*(*n*).

These complexities are summarized in Table 1. Note: key derivation is same as key computation.

Table 1 Complexities of the ACP based key management.

Terms	Comp.		Terms	Comp.
User end storage	*O*(1)		**Key computation**	*O*(*n*)
			Key derivation	*O*(*n*)
Server end storage	*O*(*n*+*m*)		***P*(*x*) generation**	$O(n^2)$
			Communication	*O*(*n*)

III. Discussions

In this section, we first discuss the anonymity issue and ACP's application to other access control models and then discuss improvement to the ACP scheme. Before going into the details, we briefly compare it with the well known Shamir's secret sharing scheme [35]. Both are based on polynomials and hide the secret/key in the polynomials. However, they are essential different. The polynomial in Shamir's scheme is kept secret but it is public in ACP. The purpose of Shamir's scheme is to prevent any single user from obtaining the secret. In contrast, ACP is intended for any of the group members to be able to obtain the secret himself. In Shamir's scheme, the secret is obtained by recovering the polynomial using polynomial interpolation which requires the number of participants be larger than the polynomial degree. In contrast, ACP just plugs a user's *SID* in the polynomial and recovers the secret key.

A. Randomized ACP and Hiding of Groups and Group Membership

One specific advantage of the ACP scheme over many existing schemes is that it can hide the group membership from outsiders (even insiders) and does not require member serialization. In many existing schemes[2], when the key(s) (after being masked) is multicast to a group of users, the information identifying these users needs to be included in the multicast packet. In addition, the users need to be ordered according to some strategy (referred to as *serialization*), so that each user knows which portion of the protected key material belongs to him and is thus able to extract the group key from that portion. These requirements not only result in more computation work (e.g., a user needs to search for his portion) and need synchronization due to the serialization but also unintentionally result in disclosures concerning the group membership information. Keeping group membership information private to outsiders may be important in some applications. Furthermore, it may be desirable or even necessary to hide the group membership from the group users themselves in some mission-critical applications, that is, a user knows that he is in the group but does not have knowledge about who are the other members of the group. Finally, it may even be necessary to hide the size of the group. Again, because of its properties, the ACP mechanism provides an efficient and elegant solution to address such requirements: the polynomial hides the group users and does not need to sort the group members. A valid user does not need to know (in fact, he cannot know if the server does not want to tell him) the membership and the order of members but he can get the group key easily by just plugging his *SID* into the polynomial. As for hiding the group size, the ACP scheme can be easily extended for this purpose by simply including some random pseudo terms in the polynomial such as:

$$A(x) = \prod_{i \in \psi} (x - f(SID_i, z)) \prod_{j=1\cdots d} (x - VID_j) \quad (6)$$

where $VID_1, \cdots, VID_d$ are random numbers in F_q, called pseudo terms, and d is a random positive integer. As a result, the degree of $P(x)$ does not indicate the number of members involved in the computation. These pseudo terms make $P(x)$ even more randomized.

It is clear that adding random terms will increase the degree of $P(x)$, thus, impacting the efficiency of the ACP scheme. However, using the hierarchical grouping and tree-based key distribution extension discussed below, the efficiency will not be affected too much. In addition, whether to add or how many random terms to be added is a trade-off between security and efficiency and should be determined based on the requirements of concrete applications.

[2] One such scheme is: numbering users sequentially, treating users' SID_is as their secret keys and encrypting the group key K by SID_is of the users in ψ respectively in sequence.

B. HAC with "Transfer Down" and "Depth-Limited Transfer" Model

The common hierarchical access model discussed above is called "transfer up" [30], that is, the access permission of a node is transferred up to its ancestors. Some other models were proposed [30, 36], two of which are: (1) "transfer down"-- the access permission of a node is transferred down to its descendants and (2) "depth-limited transfer"-- the access permission of a node is transferred either up or down but only to a certain depth. Both new models can be easily implemented by the ACP scheme. For the "transfer down" model, the $A(x)$ of a node is constructed to include the *CID*s of its descendants. For the "depth-limited transfer" model, the $A(x)$ of a node is constructed to include the *CID*s of both its ancestors and descendants which are within the limited depth from the node.

In fact, the ACP mechanism is powerful enough to adapt to random forms of interactive/access relations among users and/or resources with the great flexibility. These relations include, but are not limited to, equivalent users/resources, one-to-many, many-to-one, many-to-many, hierarchy, multiple levels, etc. For example, if a node C_i's access permission needs to be transferred to a random other node C_j, regardless of the relation and distance between the two nodes in the hierarchy, just include C_j's CID_j in the construction of $A_i(x)$. This power enables ACP to implement integration across domains: suppose C_i is in one domain and $C_{i'}$ is in another, if C_i wants (and is allowed) to access the resources in $C_{i'}$, simply put C_i's CID_i in the formation of $A_{i'}(x)$, i.e. adding the access relation from C_i to $C_{i'}$.

C. Improvement and Extension

From the above complexity analysis, it is clear that all complexities are proportional to *n*, the number of current users in the group. If *n* is large but just a single user or few users join or leave the group, $O(n)$ is not efficient. We discuss several ways to improve its efficiency. (1) As for join, the server can just generate a new key and encrypt the new key with the old group key and send it to the group. The server also encrypts the new key with the *SID* of the joining user and sends it to the joining user. (2) To improve the efficiency of computing $P(x)$, we can store and save $A(x)$ in advance. If one or a few users $U_1, \cdots, U_k$ leave, we can get the new A(x) by directly dividing $A(x)$ by $(x - f(SID_1, z)) \cdot \cdots \cdot (x - f(SID_k, z))$, thus, the complexity for generating $P(x)$ will reduce to $O(n)$. (3). For improving the efficiency of key computation and derivation, we can divide the *n* users into $k=n/l$ separate groups of *l* users each. The server forms *k* polynomials of degree *l* each. Every user can obtain the key by replacing its own *SID* to its corresponding polynomial. Thus, the complexity for key computation/derivation will reduce to $O(l)$. Next, we describe a mechanism which can improve the efficiency greatly: tree based multiple level and hierarchical grouping.

Suppose *n* is the number of all users and *m* is the size of a small group which can be managed easily and efficiently, for example, *m*=16. Then every *m* users form a first level group, so a total *n*/*m* such groups $G_{1,1},\cdots,G_{1,n/m}$ are formed. Next, every *m* first level groups form a second level group, thus, a total n/m^2 such groups $G_{2,1},\cdots,G_{2,n/m^2}$ are formed. By continuing with this strategy, finally a highest level group is formed $G_{\log_m^n,1}$. All these groups can be treated as nodes in an *m*-ary tree of height $\log_m^n$. Every group $G_{i,j}$ is associated with a group key $K_{i,j}$ and the $K_{\log_m^n,1}$ will be the group key for all users. The group keys are distributed using the ACP scheme itself. For example, $K_{1,j}$ is distributed to group $G_{1,j}$ by forming the ACP polynomial using the *SID*s of the users in its group, i.e. $P_{1,j}(x)=\prod(x-f(SID_i,z))+K_{1,j}$ where $U_i\in G_{1,j}$. The second level key $K_{2,j}$ is distributed to all users belonging to group $G_{2,j}$ by forming the ACP polynomial using the group keys of its first level groups, i.e. $P_{2,j}(x)=A_{2,j}(x)+K_{2,j}=\prod(x-f(K_{1,i},z))+K_{2,j}$ where $G_{1,i}\in G_{2,j}$. Finally, the highest level key will be distributed by forming $P_{\log_m^n,1}=\prod(x-f(K_{\log_m^n-1,i},z))+K_{\log_m^n,1}$.
Let us consider the case of a single user leaving his group. The group keys along the path from the leaf group of the leaving user to the root group need to be changed. Total $\log_m^n$ polynomials of degree *m* need to be computed and broadcast. Thus, the total polynomial generation time will be $O(m^2\log_m^n)$, the communication complexity is $O(m\log_m^n)$, and the key computation and derivation are also in $O(m\log_m^n)$. For example, suppose *m*=16, $n=2^{64}$, then the polynomial generation time, key computation time, and communication complexity are in 2048 units of time, 256 units of times, and 256 units of numbers for transmission.

It is worthy to mention that there is a simple scheme: the key *K* is encrypted with SID_i s respectively and sent to the group (see footnote 2 in the previous page). However such a scheme has some limitations compared with the ACP mechanism, such as (1) good for SGC but inflexible for HAC; (2) unable to hide group size and membership; (3) difficult to improve its efficiency, and (4) requiring for member serialization.

IV. CONCLUSIONS

In this paper, we presented an innovative concept of Access Control Polynomials. We showed how this uniform technique can accommodate various key management schemes for most security needs in various application domains. The paper has also analyzed its security and performance. From both theoretical and practical points of view, the ACP scheme is generic, flexible, efficient, dynamic, practical, invulnerable and easy to implement. Due to the space limitation, we omitted the comparison with existing schemes and illustrative examples. This could be combined with the future work in implementing and simulating the ACP technology. We are employing ACP based security functions in a real TCC application: Secure and Scalable Medical Information System for the Department of Veterans Affairs.

In summary, the main features of the new proposed ACP scheme include:

1. It is elegant, simple, easy to understand and implement.
2. It is flexible and easy to adapt to (any) different kinds of key management and (any) different kinds of access control relations.
3. It is able to enforce access control and secure group communication in any scale and any granularity.
4. It is able to implement seamless integration of heterogeneous data sources and systems without much modification of the existing components.
5. It is able to protect against any kind of attacks, not only external attacks, but also internal attacks, even when attackers and/or malicious users collude.
6. It supports highly dynamic environments; in particular, the revocation of members/resources is simple and efficient. It also supports temporary suspension of membership.
7. It does not require member serialization or synchronization and does not disclose membership.
8. Users only need to store a secret value. Furthermore, the key computation and key derivation are executed by the same efficient procedure. This makes the scheme applicable to various devices including those with low computing powers such as PDAs, sensors.
9. It is able to offer the capability of hiding the entities in groups and even group size.

10. It is able to implement flexible key management "on the fly".

In short, this novel scheme provides a holistic and seamless key management solution to foster Trusted Collaborative Computing.

ACKNOWLEDGMENT

We thank the U.S. Department of Veterans Affairs, in particular, Dr. Bradley Doebbeling and Mr. John Burke, for the support and effort with this work.

REFERENCES

[1] C. P. Pfleeger, and S. L. Pfleeger, "Security in Computing," 3rd Edition, Prentice Hall, 2003, ISBN: 0-13-035548-8.

[2] X. Zou, B. Ramamurthy, and S. Magliveras, "Secure Group Communication over Data Networks," Springer, 2004, ISBN: 0-387-22970-1.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11 – 33, 2004.

[4] G. Badishi, I. Keidar, and A. Sasson, "Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast," IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 1, pp. 45 – 61, 2006.

[5] G. Caronni, K. Waldvogel, D. Sun, and B. Plattner, "Efficient security for large and dynamic multicast groups," Proceedings of the Seventh IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 376–383, 1998.

[6] X. S. Li, Y. R. Yang, M. Gouda, and S. Lam, "Batch Rekeying for Secure Group Communications," Proc. 10th Int'l WWW Conf., pp. 525—534, 2001.

[7] D. Liu, P. Ning, and K. Sun, "Efficient self-healing group key distribution with revocation capability," ACM CCS, pp. 231—240, 2003.

[8] W. H. D. Ng, M. Howarth, Z. Sun, and H. Cruickshank, "Dynamic Balanced Key Tree Management for Secure Multicast Communications," IEEE Transactions on Computers, 56(5), pp. 577—589, 2007.

[9] G. Noubir, F. Zhu, and A. H. Chan, "Key Management for Simultaneous Join/Leave in Secure Multicast," IEEE International Symposium on Information Theory, pp. 325—330, 2002.

[10] A. Perrig and J. D. Tygar, "Secure Broadcast Communication in Wired and Wireless Networks," Kluwer Academic Publishers, 2002, pages 240, ISBN: 0-7923-7650-1.

[11] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," IEEE/ACM Transactions on Networks, 8(1):16–30, 2000.

[12] S. Mittra, "Iolus: A framework for scalable secure multicasting," J. of Computer Communication Reviews, 27(4):277—288, 1997.

[13] Y. Amir, Y. Kim, C. Nita-Rotaru, J. L. Schultz, J. Stanton, and G. Tsudik, "Secure Group Communication Using Robust Contributory Key Agreement," IEEE Transactions on Parallel Distrib. Syst., 15(5), pp. 468—480, 2004.

[14] H. Chan, V. D. Gligor, A. Perrig, and G. Muralidharan, "On the distribution and revocation of cryptographic keys in sensor networks," IEEE Transactions on Dependable and Secure Computing, vol. 2, no. 3, pp. 233 – 247, 2005.

[15] D. R. Dondeti, S. Mukherjee, and A. Samal, "DISEC: a distributed framework for scalable secure many-to-many communication," In Proceedings of fifth IEEE Symposium on Computers and Communications (ISCC 2000), pp. 693-698, 2000.

[16] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," ACM Transactions on Information and Systems Security, 7(1):60—96, 2004.

[17] M. Ramkumar, and N. Memon, "An Efficient Key Pre-Distribution Scheme for Ad-Hoc Network Security," IEEE Journal on Selected Areas in Communication., 23(3):611-621, 2005.

[18] M. Steiner, G. Tsudik, and M.Waidner, "Diffie-Hellman key distribution extended to group communication," ACM Conference on Computer and Communications Security (CCS), pp. 31—37, 1996.

[19] B. Sun, W. Trappe, Y. Sun, and K. J. R. Liu, "A time-efficient contributory key agreement scheme for secure group communications," Proceedings of IEEE International Conference on Communications, vol. 2, pp. 1159--1163, 2002.

[20] P. Adusumilli, X. Zou, and B. Ramamurthy, "DGKD: Distributed Group Key Distribution with Authentication Capability," Proceedings of the IEEE Workshop on Information Assurance and Security, pp. 476—481, 2005.

[21] Y. Desmedt, and V. Viswanathan, "Unconditionally secure dynamic conference key distribution," Proceedings of the IEEE International Symposium on Information Theory, pp. 383—383, 1998.

[22] G. H. Chiou, and W. T. Chen, "Secure broadcasting using the Secure Lock," IEEE Transactions on Software Engineering, 15(8):929—934, 1989.

[23] M. G. Gouda, C.-T. Huang, and E. N. Elnozahy, "Key trees and the security of interval multicast," Proceedings of the 22nd International Conference on Distributed Computing Systems, pp. 467—468, 2002.

[24] C. Blundo, L. A. F. Mattos, and D. R. Stinson, "Generalied beimel-chor scheme for broadcast encryption and interactive key distribution," Theoretical Computer Science, vol. 200, pp. 313-334, 1998.

[25] M. Abdalla, Y. Shavitt, and A. Wool., "Key management for restricted multicast using broadcast encryption," IEEE/ACM Transactions on Networking, 8(4):443–454, 2000.

[26] N. Kogan, Y. Shavitt, and A. Wool, "A practical revocation scheme for broadcast encryption using smart cards," Proceedings of the IEEE Symposium on Security and Privacy (SP03), pp. 225–235, 2003.

[27] A. Wool, "Key management for encrypted broadcast," ACM Trans. on Information and System Security, 3(2):107–134, 2000.

[28] S. G. Akl, and P. D. Taylor, "A cryptographic solution to the problem of access control in a hierarchy," ACM Transactions on Computer Systems, pp. 239-248, 1983.

[29] S. T. Mackinnon, P. D. Taylor, H. Meijer, and S. G. Akl, "An optimal algorithm for assigning cryptographic keys to control access in a hierarchy," IEEE Transactions on Computers, vol. 34, no. 9, pp. 797-802, 1985.

[30] M. J. Atallah, K. B. Frikken, and M. Blanton, "Dynamic and efficient key management for access hierarchies," Proceedings of the tenth ACM Conference on Computer and Communication Security (CCS' 05), pp. 190-202, 2005.

[31] G. C. Chick, and S. E. Tavares, "Flexible access control with master keys," Advances in Cryptology: CRYPTO'89 LNCS, vol. 435, pp. 316-322, 1990.

[32] M. L. Das, A. Saxena, V. P. Gulati, and D. B. Phatak, "Hierarchical key management scheme using polynomial interpolation," SIGOPS Operating Systems Review, pp. 40—47, 2005.

[33] R. S. Sandhu, and P. Samarati, "Access control: principle and practice," IEEE Communications Magazine, 32(9):40—48, 1994.

[34] M. V. Tripunitara, and N. Li, "Access control: Comparing the expressive power of access control models," Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04), pp. 62—71, 2004.

[35] A. Shamir, "How to share a secret," Communications of the ACM, 22(11):612–613, 1979.

[36] J. Crampton, "On permissions, inheritance and role hierarchies," ACM CCS'03, pp. 85-92, 2003.