GI 2004

A Hierarchical Framework for Secure Multicast Key Generation, Distribution, and Group Admission

R. Poovendran, M. S. Corson, J. S. Baras

CSHCN, Institute for Systems Research,
University of Maryland at College Park
College Park, MD 20742
e-mail: {radha, corson, baras}@isr.umd.edu

ABSTRACT

This paper presents an arbitrarily-scalable, hierarchical framework for key management in large-scale distributed systems. The proposed method makes use of a recently-proposed multiparty key generation procedure for generating the necessary session keys. This procedure is noteworthy as it is the first method that permits multiple parties—with only *limited trust* in one another—to come together and cooperatively generate shared security keys to be used for some mutual purpose; e.g. electronic commerce, distributed gaming or large-scale applications using multicast. Furthermore, it permits an increase in security to be obtained through *non-cryptographic* means, and this increase grows *more than linearly* with the number of parties involved. Overall, the proposed framework is scalable in terms of the basic security-related operations like group admission, revocation, key distribution and, more importantly, key generation.

Key words: **Internet Security, IP Multicast, Key Management, Fractional Keys**

1. Introduction

National or globally-scaled information systems may involve large numbers of participants, and it is likely that distributed commerce, conference or multicast-based systems will have a need for *scalable* multicast key generation and management technology. Given the large size of these systems, it is unlikely that they will have the same level of *physical* security throughout. Thus a scalable security infrastructure should be able to accommodate a system's *heterogeneous* security requirements. For example, this may entail generating keys with differing lifetimes and key lengths in various portions of the system.

Also, in many cases, the entities engaged in such cooperative data exchange oftentimes have only *limited or partial trust* in one another (e.g. businesses involved in electronic commerce), yet these entities need a secure shared key with which to perform secure multicast communications, and would like to obtain one without the help of (and need to pay) a trusted third party. In other words, the parties involved in the exchange, and *only* the parties involved, should be able to cooperatively generate a key without giving complete trust to *anybody*. Thus, a mechanism that permits untrusting parties to cooperatively generate a secure, shared key—without requiring the aid of a trusted third party—is desirable.

In this paper, we describe such a key generation mechanism, and show how it can be imbedded in a robust and scalable multicast key management protocol that supports heterogeneous key generation.

2. Prior Work

Some of the recent publications address issues like the distributed registration and key distribution [4,5,7], and the separation of the traditional Key Distribution Center (KDC) from the group control authority [1,2,3]. In [1,2,3,7], there is a single authority that generates and distributes the necessary keys to the whole group. At time index n, every member has access to the Group Traffic Encrypting Key ($GTEK_n$) and the Group Key Encrypting Key ($GKEK_n$) which together comprise the Group Key Packet (GKP_n). Such approaches can lead to severe security problems. If a single member is compromised in such a system, then the *whole group* has to rekey the keys or live with the compromised member. A recently-proposed approach named DiRK [7] is targeted towards the online universities and uses such a single keying scheme.

A major limitation with such key management systems is that, while they follow a distributed key management scheme, they treat all the portions of network in the same fashion; i.e. the network is *homogeneous* in terms of its security parameters. We note that, depending on the type of the network, it is possible that some portion of the network is less secure, and hence, the security parameters may have to be different for different parts of the network—security should be *heterogeneous*. For example, in a physically-less secure network portion, the key length should be longer and the key lifetime shorter than in a more secure network portion. Hence, it may become necessary to have restricted crypto transparency depending on the level of security of the group. Thus, it becomes important to be able to partition and compartmentalize the group into a collection of locally-managed clusters.

Scalable Multicast Key Management Protocol

Using these observations, we recently proposed a Scalable Multicast Key Management Protocol (SMKMP) which addressed some of these problems by incorporating Byzantine robustness and scalability [4]. The SMKMP is a multi-level, tree-shaped security framework composed of hierarchically-clustered control panels, referred to as Cluster Control Panels (CCP). There are N levels in the hierarchy. Each level i has C_i control panels or clusters, and there is only one panel ($C_1 = 1$) at the top-most level (level 1). A cluster control panel at level i is denoted CCP^i, and the j-th cluster control panel at level i is denoted CCP^i_j. There are n panel members per control panel, and the k-th panel member in CCP^i_j is denoted $P^i_{j,k}$, The lowest level of the hierarchy contains no control panels—only non-panel group members. Note that the top-most panel is sometimes referred to as the Group Control Panel (GCP), while the remaining panels are always referred to as Cluster Control Panels (CCP). Figure 1 shows the multi-level SMKMP hierarchy.

At the top level is the GCP (i.e. CCP^1_1) which consists of n $(= 4)$ members in this figure. In the next level, we have shown C_2 $(= 5)$ CCP^2s. The arrows from the CCP^1_1 (the GCP) to the CCP^2s (the second-level CCPs) indicate that each member from CCP^1_1 communicates with a *unique* member from each CCP^2. For example, using the notation in the figure, CCP^1_1 member $P^1_{1,1}$ has the trust of the CCP^2 members $P^2_{1,1}, P^2_{2,1}, \ldots, P^2_{C_2,1}$.

Comparing the CCPs with the GCP (and its generation of GKP_n), each CCP^i_j also has the permission and ability to generate a local Cluster Key Encrypting Key ($CKEK^i_{j,n+1}$) and a Cluster Traffic Encrypting Key ($CTEK^i_{j,n}$) and, hence, a Cluster Key Packet ($CKP^i_{j,n}$) that is distributed among its cluster members. When the group is large enough such that more control panels than a single GCP are necessary—which then requires at least a three-level hierarchy; in these cases, we

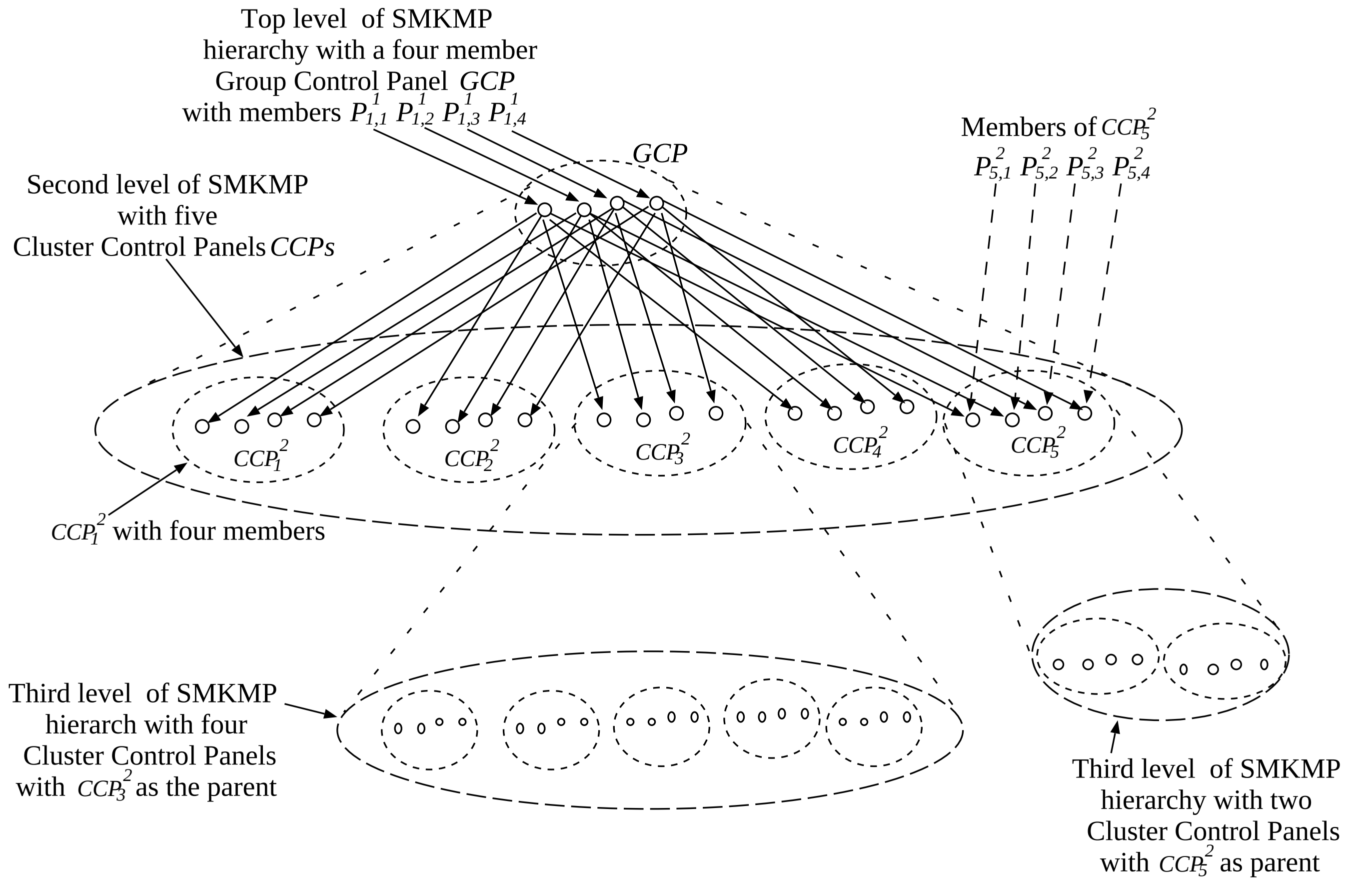

Figure 1. A multi-level SMKMP hierarchy

propose that $CKEK^i_{j,n+1} \neq CKEK^{i+1}_{k,n+1}$, $\forall j,k$ since otherwise the loss of a $CKEK^i_{j,n+1}$ creates the loss of all future keys unless the group is rekeyed.

Multiparty Key Generation Scheme

We have developed a distributed multiparty conference key generation scheme using *Fractional Keys* [5]. This method allows more than one member to participate in generating a shared key, and to automatically update the keys in a periodic manner without any assistance from an external third party. Such an approach is useful in systems where the control panel members may be from different organizations which have only partial trust in each other.

In this paper, we show how this multiparty key generation scheme can be embedded within the SMKMP to overcome some of the shortcomings of the DiRK[7], CBT[3], and GKMP[1,2].

3. Characteristics of the Multiparty Key Generation Scheme

Before giving the details of the approach, we first introduce some notation. Each panel member contributes in generating a final shared key that will be used as the secret key by the panel and the non-panel members administered by it. In doing so, each member first generates a cryptographically secure random number which we call a *Fractional Key* (FK), encapsulates it with a secure pad to generate a *Hidden Fractional Key* (HFK), and exchanges it with other panel members to generate the *Full Shared Secret Key* (FSSK). We will use Capital letters to denote the secret keys; i.e. $K_{A,B}$ will denote a secret key shared by users A, B. The notation $K_{(i,j)}$ will be used to denote the key generated or used by member i at key update time j. We will denote a *secret* S used by member i at key update instance j as $S_{(i,j)}$. The notation $\{message\}K$ is used to denote the *message* that is encrypted with the secret key K.

3.1. Properties

We note that the following properties are desirable for a multiparty key generation scheme:

1. A FK contributed by a participating member should have at least the same level of security as the FSSK.

2. A single participating member, without valid permissions, should not be able to obtain the FK of another member.

3. If a FK generating member has physically failed, the rest of the members should be able to jointly recover the FK of the failed member (this requires participation by all remaining members).

4. If a FK generating member is compromised, or removed due to administrative reasons, property 3 should still hold.

5. If a compromised or a failed node is removed, it should have no means of obtaining future FKs or FSSKs.

We note that the first property simply states that the distributed key scheme has to be such that each FK space has at least the same size as the final key space. Note that, at first glance, this property implies that if a crypto analyst picks a FK generation node for attack, the level of computational burden should be at least as much as breaking the full key. Though this is an obvious deduction, the same property leads to another interesting observation; viz, different FK spaces may be different but, when combined, they lead to a fixed level of desired security. For example, each FK bit length may vary between a multiple of one and two of the final key bit length. From the crypto analyst's point of view, it may be easier to search the FSSK space than a FK space.

The second property has to do with the need for protection of individual FK's that is desired due to the absence of a central key generation scheme. In the current scheme, every member generates a FK and securely distributes it to other members so that every panel member can derive the same FSSK. However, no individual member should be able to obtain the FK of another member. This requires that each member perform an operation to *hide* its FK such that, when all the HFKs are combined, the net result is the new key without the additional hiding components. One might think that possessing the HFK is same as possessing the actual FK. We note that such is not the case in terms of the knowledge about the FK and, even if a HFK is known, the problem of obtaining the actual FK leads to the same computational complexity for a crypto analyst. We describe the requirements of the FK concealment mechanism in the next section.

3.2. Assumptions

The following is a list of assumptions regarding the method, some of which may appear rather abstract at first glance:

1. There exist two commutative operators ω and δ which form an Abelian group when operating on the key elements.

2. It is computationally difficult to perform crypto analysis on a cryptographically-secure random key by search methods if the key length is sufficiently large.

3. The number of participants in generating the FSSK is fixed as n, the number of panel members (where n may be a function of ω and δ).

4. There is a mechanism to certify the members participating in the key generation procedure.

5. The bit length of the shared key is fixed at L, and every participating member is informed of this length.

6. Every member has the capability to generate a cryptographically-secure random number, or a fresh quantity of length at least L bits.

7. At the initialization stage, a member i participating in the key generation scheme is given a high quality random number $\alpha_{(i,1)}$ such that $\alpha_{(1,1)}\omega\alpha_{(2,1)}\cdots\omega\alpha_{(n,1)} = \theta_1$, where θ_1 is a prespecified sequence of L bits.

8. All the participating members know of the value of θ_1.

3.3. The Key Generation Algorithm

With the assumptions above, we note that the key generation scheme consists of three major parts: (a) member selection and secure initial parameter distribution, (b) FK and FSSK generation, and (c) FK recovery in the case of a member node failure. The third part is executed only when there is a node failure.

The key generation method consists of the following steps:

1. For $i = 1, ..., n$, obtain the initial parameters $\alpha_{(i,1)}$ and θ_1.

2. For $j = 1$ ...., where j is the the current end-of-key update time sequence, repeat steps 2 to 5.

3. For $i = 1, ..., n$, a member i generates a cryptographically secure random number $R_{(i,j)}$ (its FK) of length L, and generates a quantity $K_{(i,j)} = \alpha_{(i,j)}\omega R_{(i,j)}$ (its HFK).

4. Members then securely exchange these HFKs with the other participating members.

5. Each member obtains the HFKs of all others and computes the FSSK as $K_j = \theta_j \omega K_{(1,j)} \omega K_{(2,j)} \omega \ldots \omega K_{(n,j)}$.

6. If the resulting key K_j is cryptographically-insecure (such as all 1's or all zeros or first half ones or first half zeros), then any one member can perform steps 2 and 3, creating a new high quality key K_j.

7. For $i = 1, ..., n$, a member i computes $\alpha_{(i,j+1)} = K_j \omega K_{(i,j)}$. If n modulo $2 == 0$, then $\theta_{j+1} = K_j$; otherwise $\theta_{j+1} = \hat{0}$, where $\hat{0}$ is a sequence of L zero bits.

Steps 2 - 7 present the computational procedure for generating the keys at each update. At the end of step 5, we have the FSSK for the current iteration j. Note that the quantity $\alpha_{i,j+1}$ is computed such that for an outsider, obtaining $\alpha_{i,j+1}$ is as hard as obtaining the actual key K_j at any given time. We note the following features of the key scheme:

1. Although all the members have the hidden fractional key $K_{(i,j)}$ of a member i at any iteration step j, obtaining the fractional key $R_{i,j}$ involves search in the L-dimensional space. Hence, even if a fellow member becomes an attacker, the rogue member has the same amount of computational burden in obtaining the FK as a crypto analyst; i.e. trust is not unconditional.

2. By the same argument above, we note that only the member i can compute $\alpha_{i,j+1}$. Everyone else has to perform an L-dimensional search before finding the $\alpha_{i,j+1}$ which is time-varying.

3. Even if an outsider captures the packet and obtains the HFK of a single participating member, the attacker is faced with the following challenges:

   (a) The outside attacker has to find the corresponding remaining $n-1$ HFKs. Such is the case since the keys are transported in a secure manner (only the participating members have the direct access—possibly after decrypting—to the HFKs). For an outside attacker, it may be much harder to simultaneously attack and obtain these $n - 1$ parts that have limited lifetime.

(b) Having a HFK at a given time does not give any advantage to the crypto analyst in decrypting any message.

4. FKs, as well as the FSSK, are checked for standard weaknesses before being used.

5. Only the HFK or the FSSK can be attacked by performing traffic analysis. Extraction of the FK further involves an equal amount of computation on time-varying quantities.

6. If a FSSK or the HFK is known to be compromised due to an external attack, generation of a new key can be handled by the members as shown in the next section.

3.4. Dealing with Key Compromise or Node Failure

If the group members don't know that the key is compromised, then there is not much that can be done. Depending on the level of the known compromise or failure, several different procedures can be used for recovery. We will address the following types of compromise:

1. Compromise of a HFK $K_{(i,j)}$ generated by member i at iteration j.

2. Compromise of the FSSK K_j at iteration j only.

3. Failure of the member i due to an attack of either physical or crypto-type.

3.4.1. Compromise of the HFK

Although an outsider should not have access to any key generating parameters, our key scheme is such that the knowledge of a particular HFK $K_{(i,j)}$ does not directly yield the actual FK $R_{(i,j)}$ or the $\alpha_{(i,j)}$ and hence does not allow the attacker to have the direct access to the secrets of member i. In fact, if the attacker obtained only the HFK by traffic analysis, the attacker is still left with no additional knowledge. Hence, the attacker cannot use a single $K_{(i,j)}$ to capture the FSSK K_j.

3.4.2. Compromise of the FSSK

Once the K_j is known to be compromised, the following steps are followed:

1. All the members are informed using the other group-specific public keys. The manner in which this occurs is not specified.

2. All members then update their sequence counter j to $j + 1$, and using steps 2-7 of the key generation method, generate the new key K_{j+1}.

An attacker now has to perform the same amount of computation to obtain the new key K_{j+1}. In presenting the steps, we assumed that only the FSSK K_j was compromised.

3.4.3. Failure of a Node

We define the failure of node i to include both the physical failure of the node i as well as a failure due to a crypto attack on the node i and the associated loss of the keys. We note that the crypto failure of a node is much more costly from the key generation and distribution point of view. In equating the two, we assume the worst case from a security perspective and require that recovery from compromise leads to:

1. Election of a new node to replace i.

2. Enabling only the new member with an approach to recover the previous FK $R_{(i,j)}$ of member i.

3. Generation of the new FSSK K_{j+1}.

Having chosen to treat a compromised node the same as a failed node, there is still more than one possible approach to generating the new FSSK K_{j+1}. One of them is to update the key generating parameters of the whole group. However, such an approach may not be desirable if the group needs to continue to function while the compromised member is being replaced.

Another solution is to have a mechanism so that once a new member is chosen, only the new member has to update the parameters and continue the key generation procedure. Often such an approach may be more desirable, and this is the reason why we recommend using the hidden fractional key distribution at every step. Once the secret of the failed node i is extracted by the newly-elected node, the new node can compute the quantity $\alpha_{(i,j)}$, and then all the group members can update j to $j+1$ and follow the steps 2-6 in key generation method.

Such a replacement method assumes that, under certain conditions, there is a way to recover the FK $R_{i,j}$ generated by node i at iteration j without an extensive search procedure. We describe one such method in the next section.

3.4.4. Recovery of the FK of a Failed Node

The following steps are involved in recovery of the FK $R_{i,j}$ of the node i where j represents the iteration number in which the node was compromised and failed.

1. One or more members give the *HFK* $K_{(i,j)}$ of the failed node i to the newly-elected node i.
2. All the members also securely give the FSSK K_j and their HFKs used to generate K_j to the newly elected member i.
3. An entity not participating in the key generation procedure is chosen.
4. The chosen entity generates $n-1$ cryptographically secure random numbers $\beta_1, \beta_2, ..., \beta_{n-1}$ such that $\beta_1 \delta \beta_2 \delta \ldots \delta \beta_{n-1} = \gamma$.
5. For $l = 1, ..., n-1, l \neq i$, the chosen entity then securely distributes β_l and γ to member l. (i.e. no two members get the same β).
6. For $l = 1, ...n-1, l \neq i$, all the members except the newly-elected node i then compute the modified HFKs $\hat{K}_{(l,j)} = \beta_l \delta R_{(l,j)}$ and securely give these to node i.
7. Node i then combines all of them and recovers the compromised fractional key $R_{(i,j)}$ using the operation: $R_{(i,j)} = \gamma \delta \beta_1 \delta R_{(1,j)} \omega ...... \omega \beta_{n-1} \delta R_{(n-1,j)} \omega K_j$.

Once the new node i recovers the FK of the compromised node i, it can inform other contributing members to update the iteration number j to $j+1$, and then all members can execute steps 2-7 of the key generation method. Note that even though the newly-elected member recovers the compromised FK, the next key generation operation of the node i does not use the compromised FK. Hence, even if the attacker possess the FK at iteration j, it does not allow the attacker to obtain the future FKs without any computation.

3.5. A Specific Choice of Functions ω and δ

We have so far presented a class of multiparty key generation algorithms, where a given instance of the class is determined by choice of functions ω and δ. Depending on the choice of these functions, the lower bound on the number of members in a panel n may be determined. In order to present a concrete numerical example, we note that one of the possible choices for ω and δ is the modulo 2 ($\oplus$) function. In this case, we note that $n \geq 4$ is required for a non-trivial solution. For this choice of ω, if $n = 2$, it can be checked that, independent of the non-trivial bit length L of the key, it is not possible to generate a set of two initial values of $\alpha_{(1,1)}$ and $\alpha_{(2,1)}$ such $\alpha_{(1,1)}\omega\alpha_{(2,1)} = \theta$, where a member i knows only one of the two α values, but has the value of θ as well (i.e. given one value of α and θ, for $n = 2$, the other value of α is computable). So $n = 2$ is not secure.

Although $n = 3$ does not instantly expose the secret α, the following interesting attack is feasible. When a participating member becomes an attacker (i.e. a *rogue*), although that member cannot obtain the secret of the other two members directly, it can disturb the whole key generation procedure by simply giving away his own α component at any time. Once this is done, the other two members can compute each others α components. Since such an attack is member-inflicted and based on the *rogue* member's secrets, we call it a *Fractional Attack* (FA). This leads to the following lemma:

Lemma: When ω is an $\oplus$ function, independent of how non-trivial the bit-length of the key is, choosing $n = 3$ permits a *FA* attack.

Proof: Lets assume that the time instant at which one member i ($i = 1$ or 2 or 3) becomes a *rogue* is j. At this time the members have values of $\alpha_{(1,j)} = K_{(2,j)} \oplus R_{(3,j)}$, $\alpha_{(2,j)} = R_{(3,j)} \oplus R_{(1,j)}$, $\alpha_{(3,j)} = R_{(1,j)} \oplus R_{(2,j)}$. Every member also has access to the current shared key K_j and their own fractional key denoted by $R_{(l,j)}$ ($l = 1,\ 2,\ 3$). At this stage, obtaining the α component of any other member is as computationally intensive as an outside attacker trying to obtain the shared key K_j. However, if a member, say $i = 1$, is compromised and releases its secret $\alpha_{(1,j)}$, then each of the other members can use this and compute $R_{(1,j)} = \alpha_{(1,j)} \oplus K_j$. Since the shared key $K_j = R_{(1,j)} \oplus R_{(2,j)} \oplus R_{(3,j)}$, each member can now compute the other non-rogue member's FK as well. This leads to the following corollary.

Corollary: When ω is an $\oplus$ function, independent of how non-trivial the bit-length of the key, the minimum number of members to avoid a FA by a single *rogue* member for the multiparty key scheme is 4.

4. Application of the Scheme Within SMKMP

The general structure of the hierarchical key management framework is shown in Figure 1. This logical framework can be implemented on top of any multicast communications infrastructure.

We now show how the multiparty key generation scheme can be used in large-scale multicast applications where scalability in terms of the key management is an important issue. One example might be a electronic commerce application where multiple companies are involved, and they all want to be able to form a multicast network without having to completely give up their individual group keys. Another example is a conference application such as multicast-based distribution of an online university course. In such a structure, each individual university service provider can be treated as a cluster, and the combined clusters form the multicast network. Each cluster can control its own CKP_n, and all the CCPs can together be members of a single conference group. The CCPs themselves may also have sub-clusters in the overall security architecture if desired.

In what follows, we use the example of a three-level hierarchical security framework (shown in

Fig. 2) for illustrating GKP and CKP generation within SMKMP. The method is generalizable to multi-level hierarchies as well.

In this scheme, all four GCP panel members jointly generate the GKP_n and distribute it to the second-level CCPs in a distributed, secure manner. The distribution approach is such that even if the crypto analyst tries to capture the packets, only the HFKs are obtainable (and only after breaking the cryptography with which they were coded, most likely a public key scheme) thus making it much harder to quickly extract the FSSK; i.e. the FSSKs are never transmitted by any member to any other member. This is a *significant* departure from the classical methods of handing keys. As all the HFK's are transmitted securely, to obtain them—and thus to possibly obtain the FSSK—an attacker must break (i.e. decrypt) all n HFK's. Thus security increases *linearly* with panel size. Also, in the current scheme, if the number of panel members n is even, having all n HFK's is still insufficient for obtaining the FSSK as its computation is also based on $\hat{\theta}$ which is time-varying, and known only to the panel members. In this latter case, security increases more than linearly with increasing group size and, in part, through *non-cryptographic* means. We now describe how the keys are handed by GCP members to members of the CCPs.

Each GCP member $P^1_{1,i}$ $(i = 1, 2, \ldots, n)$ hands its HFK only to the CCP members $P^2_{1,i}, P^2_{2,i}, \ldots P^2_{C_2,i}$. When the time comes to update the FSSK, the members of a given CCP can securely exchange the HFKs to generate the desired FSSK.

In the rest of the paper, we assume that ω is chosen as the XOR function and we show how to create a GKP for the three-level hierarchical key management scheme. A similar procedure is performed for CKP creation at each second-level cluster.

4.1. GKEK Generation by GCP

In this procedure, all four members are securely given initial secrets by an external security manager which also issues the certificates for the four members. The exchange of initial values is denoted by:

$SM \longrightarrow P^1_{1,1}$: $\{\alpha_{(1,1)}, \theta\}K_{P^1_{1,1},SM}$

$SM \longrightarrow P^1_{1,2}$: $\{\alpha_{(2,1)}, \theta\}K_{P^1_{1,2},SM}$

$SM \longrightarrow P^1_{1,3}$: $\{\alpha_{(3,1)}, \theta\}K_{P^1_{1,3},SM}$

$SM \longrightarrow P^1_{1,4}$: $\{\alpha_{(4,1)}, \theta\}K_{P^1_{1,4},SM}$

where $\alpha_{(1,1)} \oplus \alpha_{(2,1)} \oplus \alpha_{(3,1)} \oplus \alpha_{(4,1)} = \hat{\theta}$.

The notation $K_{P,Q}$ represents the common key between entities P and Q. A member $P^1_{1,i}$ of the GCP generates the fractional $GKEK_{(P^1_{1,i},1)}$, and XORs it with the α component it received from the SM to generate the HFK as $K_{(i,1)} = \alpha_{(i,1)} \oplus GKEK_{(i,1)}$. It then encrypts the HFK with the pairwise key, signs and exchanges the HFK with a GCP member $P^1_{1,l}$ as follows ($i = 1, 2, 3, 4$, $l \neq i, l = 1, 2, 3, 4$):

$P^1_{1,i} \longrightarrow P^1_{1,l}$: $\{K_{(1,1)}\}K_{P^1_{1,i},P^1_{1,l}}$

$P^1_{1,l} \longrightarrow P^1_{1,i}$: $\{K_{(l,1)}\}K_{P^1_{1,i},P^1_{1,l}}$

Each member then combines the HFK with $\hat{\theta}$ and their own HFK to obtain the initial shared key as $GKEK_1 = GKEK_{(1,1)} \oplus GKEK_{(2,1)} \oplus GKEK_{(3,1)} \oplus GKEK_{(4,1)}$.

At an updating step j, each member $P^1_{1,i}$ (i=1, 2, 3, 4) computes the new α components as $\alpha_{(i,j+1)} = GKEK_j \oplus GKEK_{(i,j)}$. Once the pads are computed and every member $P^1_{1,i}$ generates the new FK given by $GKEK_{(i,j+1)}$, the HFK can be generated as $K_{(i,j+1)} = \alpha_{(i,j+1)} \oplus GKEK_{(i,j+1)}$. As in the case of the first step of $GKEK_1$ generation, a member $P^1_{1,i}$ and another member $P^1_{1,l}$ exchange their HFKs as follows:

$P^1_{1,i} \longrightarrow P^1_{1,l}$: $\{K_(i, j+1)\}K_{P^1_{1,i},P^1_{1,l}}$

$P^1_{1,l} \longrightarrow P^1_{1,i}$: $\{K_(l, j+1)\}K_{P^1_{1,i},P^1_{1,l}}$

After all four members have pairwise exchanged their respective HFKs, every member can compute the $GKEK_{j+1} = GKEK_{(1,j+1)} \oplus GKEK_{(2,j+1)} \oplus GKEK_{(3,j+1)} \oplus GKEK_{(4,j+1)}$.

The $GTEK_j$ can be generated in the similar fashion. Once the $GKEK_{j+1}$ and the $GTEK_j$ are created, the $GKP_j = \{GKEK_{j+1}, GTEK_j\}$ can be created.

4.2. Reception and Distribution of GKP By CCPs

As noted previously, each GCP member $P^1_{1,i}$ can choose a fixed member $P^2_{j,i}$ from a CCP^2_j and distribute the corresponding HFK. Once the $GTEK_j$ and $GKEK_{j+1}$ are generated by the members of the GCP, they are distributed to the CCPs in the following manner:

Method 1

1. Each GCP member $P^1_{1,i},\ i = 1, 2, \ldots, n$ identifies a fixed member $P^2_{j,i}$ from a given CCP^2_j.
2. The members then generate a pairwise Diffie-Helmann shared key denoted by $K_{P^1_{1,i}P^2_{j,i}}$.
3. The GCP member $P^1_{1,i}$ then sends this key to all the corresponding members in other CCPs by encrypting this key in their public key.
4. Everyone who has the correct private key then decrypts and obtains the HFK $K_{P^1_{1,i}P^2_{j,i}}$.
5. The GCP member then encrypts its share of the HFK using the DH key and multicasts $\{K_{i,j+1}\}K_{P^1_{1,i}P^2_{j,i}}$.
6. Only the members with the key $K_{P^1_{1,i}P^2_{j,i}}$ then decrypt and get the fractional keys.
7. When needed, CCP members can combine the HFKs to obtain the desired $GKEK_{j+1}$.

Method 2

1. Each GCP member $P^1_{1,i}$ identifies a member $P^2_{l,i},\ l = 1, 2, \ldots, C_2$ from every CCP.
2. The GCP member encrypts its share of the HFK using the individual public keys of the members and multicasts $\{K_{i,j+1}\}K_{P^2_{l,i}}$.
3. Member $P^2_{l,i}$ then decrypts the message and gets the HFK.
4. When needed CCP members can combine the HFKs to obtain the desired $GKEK_{j+1}$.

Comparing these two methods, the first one has the one-time overhead of generating a DH key and distributing it to the respective members from the CCPs. But once done, the encryption overhead is less for the GCP member who has to send the HFK. From the receiver's perspective, the complexity of decoding remains the same. Unless the group is very small or requires more controlled transfer of the HFK, method two is not attractive in terms of computational complexity. We presented it here for completeness in describing the possible methods of HFK transfer. We also note that only the parent panel can distribute the relevant keys. Peer CCPs are not certified to distribute the keys among themselves. Hence, in the second level of a three-level SMKMP, no CCP can distribute the $GTEK_j$ or $GKEK_j$ to another CCP. Only the GCP performs that operation. Every CCP can, of course, generate and distribute its $CKEK_j$ and $CTEK_j$ among its local members. Note that we can control cluster formation so as to make clusters relatively small so that security administration remains highly manageable and rekeying a cluster does not involve a large number of group members.

4.3. Key Management During Cluster Splitting

Handling cluster splitting is one of the main issues in large self-organizing hierarchical networks [8]. We present an approach for handling keys when a cluster splits. There are two possible mechanisms at this level: (a) if the group is multi-level, the current CCP becomes the parent of the split CCP, (b) if the group is three-level, the current CCP becomes a peer to the new CCP due to split.

4.3.1. Three-level SMKMP

If there are only three levels of hierarchy—the first level consisting of the GCP, the second level consisting of the CCPs, and the third level consisting only of non-panel members, then all the CCPs are peers. If such a structure exists, then only the GCP has the authority to certify and distribute the HFKs to the CCPs. The key setup for the new CCP by the GCP proceeds in the following fashion:

1. Each member of the GCP selects one member from the eligible members to form the new CCP.

2. The GCP issues individual certificates to the new CCP members authorizing them to perform group admission and other security-related tasks, including CKP generation.

3. The GCP then announces the new CCP information on the multicast channel.

4. Each GCP member then securely hands the DH key to be used in handing the HFK to the new CCP member. This can be done by encrypting the DH key with the public key of a given member.

4.3.2. Multi-level SMKMP

In an N-level hierarchical framework where the CCP members are authorized to create CCPs, when the cluster splits, the following steps are performed:

1. Each member of the parent CCP selects one member from the eligible members to form the new CCP.

2. The CCP issues the individual certificates to new CCP members authorizing them to perform group admission and other security-related tasks, including CKP generation.

3. The CCP then announces the new CCP information on the multicast channel.

4. Every CCP member first identifies one of the new CCP members, and securely hands it the DH key.

Once the procedure is completed, the newly-elected CCP is ready to generate the CKP and also receive the new $GKEK_{j+1}$, and the $GTEK_j$ from the GCP, if need be.

5. Conclusions

We presented a new distributed key generation scheme that provides additional non-cryptographic security in key distribution. Our scheme is readily-scalable and does not rely on assumptions such as the difficulty of computing logarithmic operations in discrete domains. We showed how to use our scheme in the recently-proposed key management protocol SMKMP. We addressed a set of important issues such as key distribution among the control panels, key management during the cluster splitting and key recovery due to a participating node failure etc. We noted that by introducing the concept of *fractional keys*, we permit mutually-suspicious (i.e. untrusting) entities to generate and recover keys without having an external key distribution/recovery center. Our method is directly applicable to large-scale, distributed multiparty systems using multicast technology.

References

[1] H. Harney C. Muckenhirn, "GKMP Architecture", RFC 2093, July 1997.

[2] H. Harney, C. Muckenhirn, "GKMP Specification", RFC 2094, July 1997.

[3] A. Ballardie, "Scalable Multicast Key Distribution", RFC 1949, May 1996.

[4] R. Poovendran, S. Ahmed, S. Corson, J. Baras, "A Scalable Extension of Group Key Management Protocol", Proceedings of the *2nd Annual ATIRP Conference*, pp. 187 -191, Feb. 2-6, 1998, Maryland.

[5] R. Poovendran, S. Corson, J. Baras, "A Shared Key Generation Procedure Using Fractional Keys", submitted to *IEEE Milcom98*, Boston MA.

[6] M. Bellare and S. Micali, "Non-Interactive Oblivious Transfer and Applications", *Advances in Cryptology - Crypto '89*, Springer-Verlag, 547-557.

[7] R. Oppliger and A. Albanese, "Distributed Registration and Key Distribution(DiRK)", Proceedings of the *12th International Conference on Information Security (IFIP SEC'96)*, Island of Samos (Greece), pp. 199-208, May 21-24, 1996, Chapman and Hall, London.

[8] K. Kasera and R. Ramanathan, "A Location Management Protocol for Hierarchically Organized Multihop Mobile Wireless Networks", Proceedings of *IEEE ICUPC* Sandiego, 1997.

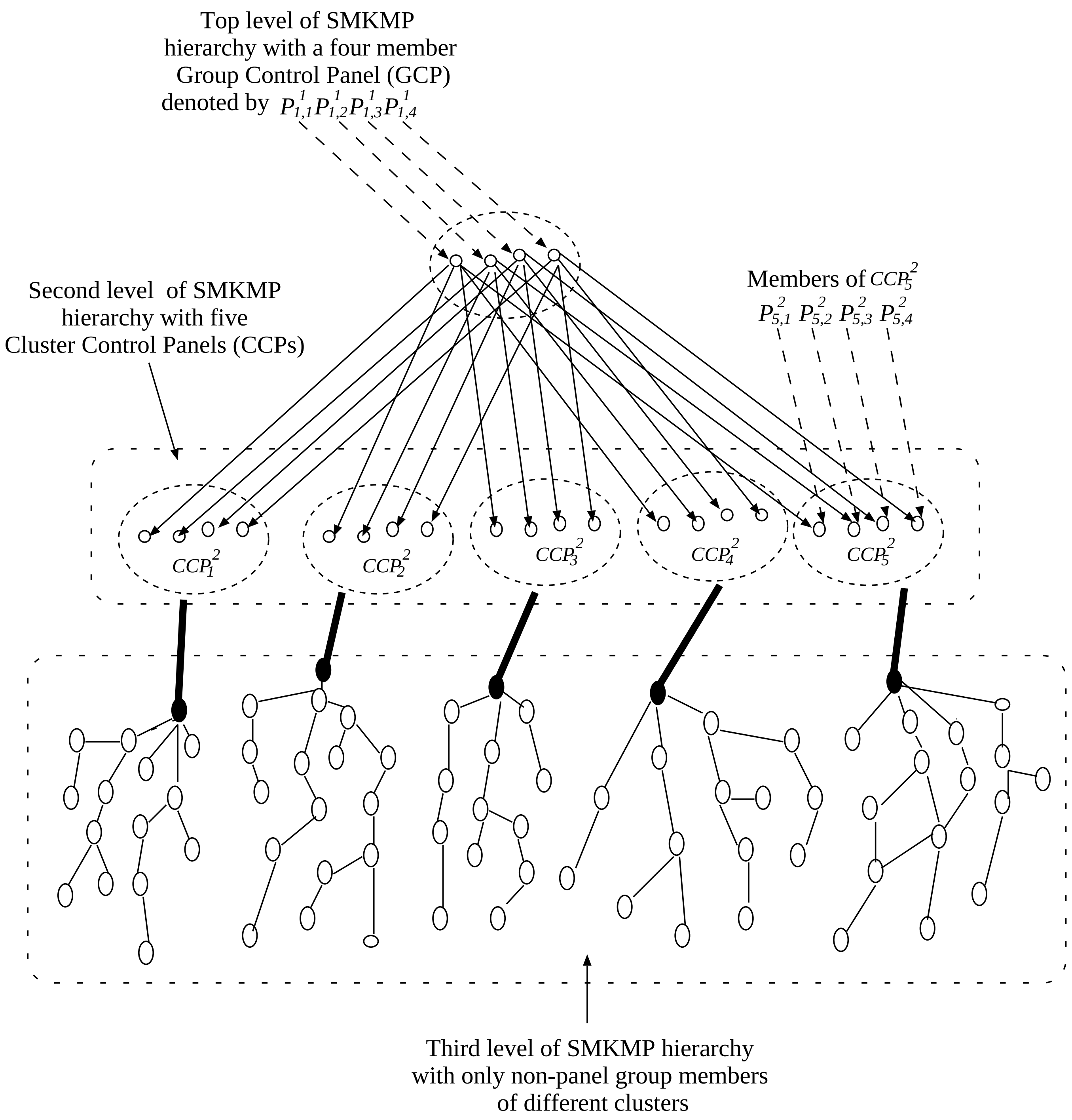

Figure 2. A three-level SMKMP hierarchy