CHAPTER 2
STATE OF ART

2.1 INTRODUCTION

In this section the various research works and techniques that are in use in the broad domain of key management for secure multicasting is discussed briefly. In any group communication, scalability and security were the two major concerns. A session which involves frequent changes in membership would become more difficult to administrate. This has resulted in the probe for more scalable and practical solutions for the group communication problem, thereby it leads to the production of smaller messages utilizing fewer resources on the Group controller’s machine. The varieties of approaches available to secure multicasting for establishing group communication are broadly classified into three different categories.

- Centralized approaches
- Distributed approaches
- Distributed subgroup approaches

In the next section the various techniques under centralized scheme are dealt in detail.

2.2 CENTRALIZED APPROACHES

In centralized approaches, there would be a central server controlling the whole group. This group controller would not rely on any auxiliary entity to perform access control and key distribution. Here the hitch would lie with the central server which becomes a single point of failure. If the group controller is more reliable, then the centralized system is the best method for supporting large member groups. The following popular conventional centralized approaches are discussed in the following sections.

(i) Group Key Management Protocol
(ii) Hierarchical Binary Tree
(iii) One-way Function Tree
(iv) Key graph and
(v) Centralized Flat Key Management
2.2.1 Group Key Management Protocol

The group key management protocol (Harney H and Muckenhirn C (1997)) would enable the creation and maintenance of a group key. In this approach the Group Controller (GC) would create a Group Key Packet (GKP) that contains a Group Traffic Encryption Key (GTEK) and a Group Key Encryption Key (GKEK). When a new member wants to join the group the GC sends him a copy of the GKP. When a re-keying is needed, the GC would generate a new GKP and encrypt it with the current GKEK. In spite of this as the GKEK is known to all the members, there is no way to ensure forward secrecy when a member leaves the group.

2.2.2 Hierarchical Binary Tree

Wallner et al (1999) proposed the use of a Hierarchical Binary Tree (HBT). In this approach there is a group controller which would maintain a tree of keys. The nodes of the tree hold Key Encryption Keys (KEKs) while the leaves of the tree correspond to group members and each leaf holds a KEK associated with that member. Each member would receive and maintain a copy of the KEK associated to its leaf and the KEKs corresponding to each node in the path from its parent to the root. The key held by the root of the tree is the group key. For a balanced tree, each member stores \( \log_2 n \) keys, where ‘n’ is the number of members. For example the balanced tree shown in Figure 2.1, member U1 knows K1, K12, K14 and K. Every joining member is accommodated in a leaf of the tree. All KEKs in the nodes that lie in the path between the new leaf’s parent and the root are compromised (to ensure backward secrecy) and have to be changed. A re-key message is generated containing each of the new KEKs which are encrypted with the corresponding KEKs of the node’s children. The size of the message produced will be at most \( O(2 \log_2 n) \). The Figure 2.2 shows an example of the KEKs being affected. The new member U3 received a secret key K3 and its leaf was attached to the node K34. The KEKs held by nodes K34, K14 and K, which are the nodes in the path from K3 to K, are compromised. New KEKs (K’34, K’14 and K’) were generated for the affected nodes. Finally the KEKs are encrypted with each of its corresponding KEKs of the node’s children (K’34...
is encrypted with $K_3$ and $K_4$; $K_{14}$ is encrypted with $K_{12}$ and $K_{34}$; and $K$ is encrypted with $K_{14}$ and $K_{58}$). The size of a re-keying message for a balanced tree has at most $O(2 \log_2 n)$ keys.

When a member departs from the group, the procedure discussed above is followed (caronni G et al 1999). In this case, the leaving member’s parent node’s KEK and all KEKs held by nodes in the path to the root are compromised (To ensure forward confidentiality) and hence have to be changed. A re-key message is generated containing each of the new KEKs encrypted with its respective node’s children KEK but the KEK of the parent to the leaving member’s leaf should be encrypted only with the KEK that are held by the remaining member’s leaf (i.e. with the sibling of the leaving node). As the key held by the leaving member was not used to encrypt any new KEK, and all its known KEK’s were changed, the evicted member is no longer able to access the group messages. Figure 2.3 presents what happens when a member leaves. Member $U_4$ is leaving the group and the KEKs $K_{34}$, $K_{14}$ and $K$ are compromised. KEKs $K’_{34}$, $K’_{14}$, and $K’$ are generated and encrypted with each of the corresponding children’s KEKs. An exception was made for the $K’_{34}$. This KEK was encrypted only with $K_3$ which is the remaining member’s key of $K_{34}$.

![Figure 2.1: KEKs affected when a member joins the tree](image-url)
Figure 2.2: Necessary encryptions needed when a member joins the tree in basic HBT

{x}k, means x has been encrypted with k

Figure 2.3: Necessary encryptions needed when a member is removed from basic HBT

{x}k, means x has been encrypted with k
2.2.3 One–way Function Tree

Yet another centralized group key management scheme is One Way Function Tree (OFT) approach. OFT was developed (Balenson et al 2000) at Network Associates Laboratories as part of the DARPA funded Dynamic Cryptographic Context Management (DCCM) and it was the first of the Hierarchical method to achieve such an approximate halving of broadcast size, an idea on which ensuing algorithms have been built.

A Special one-way function (Sherman A T et al 2003) was used to compute a tree of keys. The keys are computed up the tree, from the leaves to the root. This method reduces re-keying broadcast to about \((\log n)\) keys, where \(n\) is the number of group members.

The Group manager would maintain a binary key tree, each node \(v\) would have two cryptographic values viz., a node secret \(x_v\) and a node key \(k_v\). The node secrets are functionally related by means of a special one-way function \(f\). It will be referred to the value \(f(x_v)\) as the” blinded node secrect” of the node \(v\). The node secrect is blinded in the sense that a computationally limited adversary can know \(f(x_v)\) and yet cannot find \(x_v\). Similarly for each node the node key is computed from the node secrect using a special one way function \(g\); thus, \(k_v = g(x_v)\). The manager uses a symmetric encryption function \(E\) to communicate securely with subsets of group members. During rekeying operations, the manager would communicate with the group members through broadcast messages.

2.2.3.1 Structure of an OFT

The OFT key tree is a particular type of Binary tree, in which each interior node has exactly two children. Every leaf of the tree is associated with group member, and the node secret of the root is the common group key. Before the group key is computed a randomly chosen node secret is assigned to each member. In addition to initialization, for each member the manager establishes a separate “shared key” which
is known only to the manager and the member. For any interior node $v$ in the OFT key tree, the node secret $x_v$ of $v$ is defined by $x_v = f(x_L) \oplus f(x_R)$, where $L$ and $R$ denote left and right children of $v$.

![Figure 2.4: Structure of OFT Tree](image)

### 2.2.3.2 Group Initialization

The three steps in the initialization of a group are given below:

1. Each member establishes a “shared key” known only by the member and the group manager (Group induction). This key establishment can be accomplished with any pair wise authenticated key exchange protocol, such as Internet Key Exchange protocol (IKE).

2. The manager creates the OFT key tree and assigns initial member to its leaves. A variety of choices are possible for creating the OFT tree and numbering its nodes.
   - i) Trees can be allowed to grow and shrink dynamically as members are added and evicted.
   - ii) Large complete tree can be allocated, the nodes are numbered using in-order scheme.
3. Initial group key is computed. The members compute the initial group key after receiving crucial information broadcast from the manager. To compute the group key, each member needs to compute the blinded node secret of each sibling node along the path from the member to the root. Each member also computes the node keys along the path to the root, because these node keys are needed to decrypt relevant message parts from the manager’s broadcast.

2.2.3.3 Inserting a new member into an OFT key tree

![Figure 2.5: Before inserting a new member into an OFT Tree](image1)

![Figure 2.6: After inserting a new member into an OFT Tree](image2)
For each node v, $x_v$ is the node secret of v, $k_v$ is the node key of v, and $f(x_v)$ is the blinded node secret of v. Via a unicast, the newly added member M is informed of the blinded node secrets of its ancestral siblings v10, v4, v3. It also computes the new unblinded keys of its ancestors v5, v2, v1. Via multicast, the other relevant members are informed about the blinded node secrets present at the nodes along the new member’s path to the root. To accomplish this task, the manager broadcasts the following three encrypted messages: $E_{k_{10}}[f(x_{11})]$, $E_{k_4}[f(x_5)]$ and $E_{k_3}[f(x_2)]$, where E is an encryption function and f is a special one way function. The structure of OFT, before and after a member insertion is illustrated in Figure 2.5 and 2.6 respectively.

### 2.2.3.4 Evicting a member from an OFT key tree

![Figure 2.7: Before evicting a member from an OFT Tree](image1)

![Figure 2.8: After evicting a member from an OFT Tree](image2)
When member M is evicted, member M’s sibling (Member Z) is assigned a new node secret, which in turn alters the node secrets of its ancestors v2, v1. Via multicast, the other members who need to know are informed of the new blinded node secrets of nodes v5, v2. To accomplish this task, the manager broadcasts two encrypted messages \(E_{k1}[f(x_5)]\), \(E_{k3}[f(x_2)]\), where E is an encryption function and f is a special one way function. The structure of the OFT, before and after a member eviction is illustrated in Figure 2.7 and 2.8 respectively.

OFT achieves a smaller broadcast through its bottom up approach. OFT requires significantly fewer random bits. (The manager must generate only one new random key, to add one member). This algorithm is appropriate for an online system. Since, OFT distribute the computational cost of re-keying among the whole group, the burden on the group manager is comparable to that of a group member. OFT method has the option of allowing group members to contribute entropy to the group key. Even though the OFT achieves perfect forward security and perfect backward security, the computational complexity is too high.

The performance of OFT also depends on the height of the key tree. If dynamic trees are used, to minimize h, new members can be added as close to the root as possible. If eviction occurs, the resulting tree may be “unbalanced”, that is, its height may be much greater than \(\log n\). A rebalancing operation has to be performed.

Since OFT is intended for very large groups, the amount of Storage allocated per node in the key tree significantly affects the total amount of memory used by the manager. This issue can be referred to as manager’s node storage explosion. It is important to verify whether a group member has the correct group key. A member might compute the wrong group key for two reasons.

1. A member might never receive some key-update messages.

2. In the sequence of key update messages, the member might not have any knowledge about the additional key-update messages that are yet to come.
These situations are complicated by the unreliable nature of the multicast environment.

### 2.2.4 Keygraph

A keygraph (Wong C K et al 2000) is a directed acyclic graph \( G \) with two types of nodes i.e., u-nodes representing users and k-nodes representing keys. Each u-node has one or more incoming edges. If a k-node has only incoming edges and no outgoing edges, then this k-node is called a root.

The key graph \( G \) represents a secure group \((U, K, R)\) which has the following features:

- Where \( U \) - Set of User nodes
- \( K \) - Set of Key nodes
- \( R \) - Set containing all the possible directed paths from the leaf nodes.
- One-to-one correspondence between \( U \) and the set of u-nodes in \( G \).
- One-to-one correspondence between \( K \) and the set of k-nodes in \( G \).
- \((u, k)\) is in \( R \) if and only if \( G \) has a directed path from the u-node to k node.

![Figure 2.9: Simple Key graph](image)
As an example, the key graph in Figure 2.9 specifies the following

\[ U = \{ u_1, u_2, u_3, u_4 \} \]

\[ K = \{ k_1, k_2, k_3, k_4, k_{12}, k_{234}, k_{1234} \} \]

\[ R = \{ (u_1, k_1), (u_1, k_{12}), (u_1, k_{1234}), (u_2, k_2), (u_2, k_{12}), (u_2, k_{234}), (u_2, k_{1234}), (u_3, k_3), (u_3, k_{234}), (u_3, k_{1234}), (u_4, k_4), (u_4, k_{234}), (u_4, k_{1234}) \} \]

Associated with each secure group \((U, K, R)\) are two functions, \text{keyset()} and \text{userset()} which are defined as follows:

\[
\text{keyset}(u) = \{ k \mid (u, k) \in R \}
\]

\[
\text{userset}(k) = \{ u \mid (u, k) \in R \}
\]

where Keyset\((u)\)-is the set of keys that are held by user \(u\) in \(U\), and Userset\((k)\)-is the set of users hold by \(k\) in \(K\). For example in Figure 2.9 \text{keyset}(u_4) = \{k_4, k_{234}, k_{1234}\} and Userset(k_{234}) = \{u_2, u_3, u_4\}.

2.2.4.1 Join Operation

After granting a join request for \(u\), the server creates a new \(u\)-node for user \(u\) and a new \(k\)-node for its individual key \(k_u\). Then it finds an existing \(k\)-node (referred as the joining point for this join request) in the key tree and attaches \(k\)-node \(k_u\) to the joining point as its child. For example, suppose \(u_9\) is granted to join as shown in Figure 2.10, the joining point’s key is to be changed from \(k_{78}\) to \(k_{789}\) and the group key is also to be changed from \(k_{1-8}\) to \(k_{1-9}\). After updating the appropriate keys, the keys are distributed using anyone of the following three methods.

(i) User oriented Rekeying

(ii) Key oriented Rekeying

(iii) Group oriented Rekeying

The following sections explain each of the above methods in detail.
Method 1: User-Oriented Rekeying

For each user, the server constructs a rekey message that contains the new keys needed by the user and encrypts them using a key held by the user. For example in Figure 2.10 for the user $u_9$ to join, the server needs to send the following three rekey messages

$$S \rightarrow \{u_1 \ldots u_6\} : \{k_1-9\}k_1-8$$
$$S \rightarrow \{u_7, u_8\} : \{k_1-9, k_789\}k_78$$
$$S \rightarrow \{u_9\} : \{k_1-9, k_789\}k_9$$

![Figure 2.10: Key trees join and leave](image-url)
Note that users u1 to u6 need to get the new group key k 1-9. For each k-node x whose key has been changed, from k to k’, the server constructs a rekey message by encrypting the new keys of k-node x and all its ancestors (up to the root) by the old key k. This approach needs h rekey messages and the encryption cost for the server is \((h(h+1)-1)/2\). Here ‘h’ denotes the height of the tree i.e. the length (in number of edges) of the longest directed path in the tree.

**Method 2: Key-Oriented Rekeying**

In this method, each new key is encrypted individually. For each k-node x whose key has been changed, from k to k’, the server constructs two rekey messages.

First, the server encrypts the new key k’ with the old key k and sends it to \(\text{userset}(k)\), which is the set of users that share k. All of the original users that need the new key k’ can get it from this rekey message. The other rekey message contains the new k’ encrypted by the individual key of the joining user and is sent to the joining user.

For example, for user u9 to join in Figure 2.10, server s needs to send the following four rekey messages.

\[
\begin{align*}
S & \rightarrow \{u1...u8\} : \{k1-9\}k1-8 \\
S & \rightarrow u9 : \{k1-9\}k9 \\
S & \rightarrow \{u7, u8\} : \{k789\}k78 \\
S & \rightarrow u9 : \{k789\}k9
\end{align*}
\]

This approach needs h rekey messages and the encryption cost for the server is \(2(h-1)\).

**Method 3: Group-Oriented Rekeying**

In the case of Group-Oriented rekeying (Wong C K 1998), the server used to construct a single rekey message containing all new keys. This rekey message is then
multicast to entire group. Group-oriented rekeying method has the following advantages over key-oriented and user-oriented rekeying.

1) Multicast can be used instead of unicast or subgroup multicast.
2) With fewer rekeying messages, the server’s overhead is reduced.
3) The total number of bytes transmitted by the server per join/leave request is much less than those of key-oriented and user-oriented rekeying which duplicate information in rekey messages.

For example, for user u9 to join in Figure 2.10, server s needs to send the following two rekey messages; one is multicast to the group and the other is unicast to the joining user:

\[
\begin{align*}
S &\rightarrow \{u1\ldots u8\} : \{k1\ldots 9\}k1-8, \{k789\}k78 \\
S &\rightarrow u9 : \{k1-9, k789\}k9
\end{align*}
\]

This approach reduces the number of rekey messages to one multicast message and one unicast message and hence maintaining the encryption cost at 2 (h-1).

**2.2.4.2 Leaving A Key Tree Graph**

After granting a leave request from user u, server updates the key graph by deleting the u-node for user u and the k-node for its individual key from the key graph. The parent of the k-node for its individual key is called the leaving point.

For example, suppose u9 is granted to leave as shown in Figure 2.10. The leaving point is k789 and the key for this k-node is to be changed to k78. Moreover, the group key is also to be changed from k1-9 to k1-8 and the users’ u7 and u8 need to know the new group key k1-8 and the new subgroup key k78.

To securely distribute the new keys to the users after a leave, as in the case of join operation anyone of the following three rekeying strategies can be used.
**Method 1: User-Oriented Rekeying**

In this approach, each user gets a rekey message in which all the new keys it needs are encrypted using a key it holds. For example, as shown in Figure 2.10, for user u9 to leave the server needs to send the following four rekey messages.

- \( s \rightarrow \{u_1, u_2, u_3\} : \{k_{1-8}\} k_{123} \)
- \( s \rightarrow \{u_4, u_5, u_6\} : \{k_{1-8}\} k_{456} \)
- \( s \rightarrow \{u_7\} : \{k_{1-8}, k_{78}\} k_7 \)
- \( s \rightarrow \{u_8\} : \{k_{1-8}, k_{78}\} k_8 \)

This approach requires \((d-1)(h-1)\) rekey messages and the encryption cost for the server is given by \((d-1)(1+2+...+h-1) = \frac{(d-1)(h)(h-1)}{2}\). Here ‘d’ denotes the degree i.e. the maximum number of incoming edges of a node in the tree.

**Method 2: Key-Oriented Rekeying**

In this approach, each new key is encrypted individually. For example, as shown in Figure 2.10, for user u9 to leave, the server needs to send the following four rekey messages.

- \( S \rightarrow \{u_1, u_2, u_3\} : \{k_{1-8}\} k_{123} \)
- \( s \rightarrow \{u_4, u_5, u_6\} : \{k_{1-8}\} k_{456} \)
- \( s \rightarrow \{u_7\} : \{k_{1-8}, k_{78}\} k_7 \)
- \( s \rightarrow \{u_8\} : \{k_{1-8}, k_{78}\} k_8 \)

Note that by storing encrypted new keys for use in different rekey messages, the encryption cost of this approach is \(d\) \((h-1)\), which is much less than that of user-oriented rekeying. The number of rekeying messages is \((d-1)(h-1)\), same as user oriented rekeying.
Method 3: Group-Oriented Rekeying

A single rekey message is constructed containing all new keys. For example, for the user u9 to leave the secure group in Figure 2.10, the server needs to send the following rekey message

Let L0 denote \( \{k_{1-8}\} k_{123}, \{k_{1-8}\} k_{456}, \{k_{1-8}\} k_{78} \)
Let L1 denote \( \{k_{78}\} k_{7}, \{k_{78}\} k_{8} \)
S \rightarrow \{u_{1}…u_{8}\} : L_0, L_1.

This approach uses only one rekey message which is multicasted to entire group, and the encryption cost is \( d(h-1) \), where \( d \) is the average degree of a k-node.

The comparison of the above described three methods with respect to server processing time per request versus group size and server processing time per join versus key tree degree are shown in Figure 2.11 and Figure 2.12 respectively.

![Figure 2.11: Server processing time per request versus group size (Key tree degree 4)](image-url)
2.2.5 Centralized Flat Key Management

The Swiss Federal Institute of Technology Group extended their own solution by proposing to change the hierarchical tree for a flat table with the effect of decreasing the number of keys held by the Group Manager. The group manager assigns binary IDs to members of the multicast group. It then generates 2W number of KEKs, where W represents the length of an ID. There is a KEK corresponding to each binary value of bit positions in the IDs. Each member receives W number of KEKs corresponding to the bit values in its ID. The group manager also distributes a traffic encryption key (TEK) to all the members (Waldvogel et al 1999). This scheme is called Centralized Flat Key Management (CFKM). There are only \((2\log n)+1\) keys in the group. When a host joins the group it is assigned a binary ID and the sender rekeys all the keys \((\log n)\) corresponding to that ID. When a member leaves, the sender changes the TEK and sends it in a message that can be decrypted only by the members with IDs differing in at least a single bit compared to the departing member’s ID. The new TEK and old KEKs are used to encrypt new KEKs before sending them to the members of the group. CFKM also suggests mechanisms to join or leave groups in order to help reduce the encryption and message passing complexity of the protocol. This is often referred to as omitted key approach. This is illustrated in Figure 2.11.
Since CFKM requires the TEK to be modified each time a member joins/leaves the group, it suffers from 1 affects n scalability problem and also this protocol suffers from collusion attacks.

2.2.6 Boolean Function Minimization

Chang I, R.Engel, et. Al (1999) proposed a scheme which had the same properties as the flat table with an optimization for the number of messages needed for re-keying the group based on Boolean function minimization technique. This focuses on the problem of aggregating key updates by cumulative member removal. The main advantage is the reduction in message complexity by removing multiple group members simultaneously.

In this approach, each user has a unique user ID (UID) which is a binary string of length n. A UID can be written as Xn-1, Xn-2,…,X0 where Xi can be either 0 or 1. The length of UID depends upon the size of the multicast group. For example in a group with 1024 members a 10 bit UID is to be used. Initially the group controller distributes the session key to all the members. If any user leaves the group, the session key is changed by the group controller in order to ensure the forward secrecy and is
transmitted to all the remaining members by making use of complementing the UID of the departed member.

Consider the group contains 8 members. Suppose C5 (101) has to be removed from the group. The new session key, SKnew is generated and it is transmitted to the remaining (7) members. The new session key is encrypted by the” complementary” to the ones of the departing member and it is transmitted using 3 i.e., \( O (\log N) = O (\log 8) = 3 \) messages to the remaining ones. They are \( \{SKnew\}K’0 \), \( \{SKnew\}K1 \) and \( \{SKnew\}K’2 \) respectively where \( \{L\}M \) means string L is encrypted with key M. Hence, the departed member C5 cannot decrypt the messages.

This scheme discusses only about leave operation and does not discuss about join operation and also this is susceptible to collusion attacks by two or more members.

### 2.3 DISTRIBUTED APPROACH

Basically in a distributed approach, there is no centralized server and the group key is to be formed from the contribution of all members. The Diffie-Hellman algorithm is the base for all the key management techniques used under this approach (Stiener et al 1996). This algorithm relied upon for its effectiveness on the difficulty of computing discrete logarithms. For any integer ‘b’ and a primitive root ‘a’ of prime number ‘p’, a unique exponent ‘i’ can be found such that

\[
b = a^i \mod p, \text{ where } 0 \leq i \leq (p-1).
\]

The exponent ‘i’ is referred to as the discrete logarithm.

A primitive root of a prime number ‘p’ is one whose powers generate all the integers from 1 to p-1. i.e. if ‘a’ is a primitive root of a prime number ‘p’, then the values a \( \mod p \), a^2 \( \mod p \), …, a^\(p-1\) \( \mod p \) are distinct and consist of the integers from 1 through p-1 in some permutation. For example the calculation of primitive roots of the prime number 19 is illustrated by the following Table 2.1
Table 2.1: Powers of Integers, Modulo 19

<table>
<thead>
<tr>
<th>a</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
<th>a5</th>
<th>a6</th>
<th>a7</th>
<th>a8</th>
<th>a9</th>
<th>A10</th>
<th>a11</th>
<th>a12</th>
<th>a13</th>
<th>a14</th>
<th>a15</th>
<th>a16</th>
<th>a17</th>
<th>a18</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>13</td>
<td>7</td>
<td>14</td>
<td>9</td>
<td>18</td>
<td>17</td>
<td>15</td>
<td>11</td>
<td>3</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>15</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>18</td>
<td>16</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>4</td>
<td>12</td>
<td>17</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>7</td>
<td>9</td>
<td>17</td>
<td>11</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>7</td>
<td>9</td>
<td>17</td>
<td>11</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>9</td>
<td>7</td>
<td>16</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>9</td>
<td>7</td>
<td>16</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>9</td>
<td>16</td>
<td>1</td>
<td>6</td>
<td>17</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>9</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>18</td>
<td>11</td>
<td>12</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>18</td>
<td>11</td>
<td>12</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>18</td>
<td>11</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>16</td>
<td>11</td>
<td>4</td>
<td>17</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>16</td>
<td>11</td>
<td>4</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>12</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>15</td>
<td>17</td>
<td>18</td>
<td>9</td>
<td>14</td>
<td>7</td>
<td>13</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>18</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>12</td>
<td>11</td>
<td>18</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>12</td>
<td>11</td>
<td>18</td>
<td>7</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
<td>12</td>
<td>4</td>
<td>14</td>
<td>11</td>
<td>10</td>
<td>16</td>
<td>18</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>15</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>8</td>
<td>17</td>
<td>10</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>18</td>
<td>5</td>
<td>13</td>
<td>11</td>
<td>2</td>
<td>9</td>
<td>12</td>
<td>16</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>8</td>
<td>9</td>
<td>2</td>
<td>11</td>
<td>13</td>
<td>5</td>
<td>18</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td>17</td>
<td>8</td>
<td>6</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td>11</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>17</td>
<td>6</td>
<td>1</td>
<td>16</td>
<td>9</td>
<td>11</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>17</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>11</td>
<td>16</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>1</td>
<td>17</td>
<td>4</td>
<td>11</td>
<td>16</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>1</td>
</tr>
</tbody>
</table>

From the above Table 1, it is inferred that the primitive roots of 19 are 2, 3, 10, 13, 14 and 15 because the sequences generated by various powers of the primitive roots (21, 22 …218 31, 32 …318 etc.) are of length 18 and values are in the range of 1 to 18 (i.e. 1 to p-1) without any repetition. The significance of discrete logarithm as a hard problem in key exchange is explained in the next section.

2.3.1 Diffie –Hellman Key Exchange

The following example illustrates (William Stallings 2000) how a common key is formed by users A and B by exchanging the partial keys between them. In this scheme two publicly known numbers are used, namely q and α an integer that is a primitive root of q. User A selects a random integer XA<q and computes YA=α XA mod q. Similarly user B selects a random integer XB<q and computes YB= α XB mod q. Each side keeps the X value private and makes the Y value available to the other side. User A computes the key as K= (YB) XA mod q and user B computes the key as K= (YA) XB mod q. These two calculations produce identical result which is nothing but a common key.
\[ K = (YB) XA \mod q \quad - \quad \text{(1)} \]
\[ = (\alpha XB \mod q) XA \mod q \]
\[ = (\alpha XB) XA \mod q \quad \text{by the rules of modular arithmetic} \]
\[ = \alpha XB XA \mod q \]
\[ = (\alpha XA) XB \mod q \]
\[ = (\alpha XA \mod q) XB \mod q \]
\[ = (YA) XB \mod q. \quad - \quad \text{(2)} \]

From the equations (1) and (2) it is evident that users A and B arrived at their common key in a secret manner.

**2.3.2 CLIQUES Protocol**

The two party Diffie-Hellman key exchange is extended for a group and is referred as CLIQUES protocol by Steiner et al (1998) A clique is an extension of Diffie-Hellman key agreement protocol. In DH key agreement protocol, two parties agree on a common key. In cliques, instead of two entities, the group agrees on a common key. All the members contribute to generate the group key. The setup time is linear since all members are involved in sequence to generate their group keys. It provides an Application-Programming Interface (API) for secure group communication. This API enables all the group operations. Here all the members can perform access control. It can handle group dynamism effectively and can handle fault tolerant issues. Group controllers are necessary for membership management and group key refreshes are based on the local polices. There are basically two key management operations namely initial key agreement (IKA) and auxiliary key agreement (AKA). It supports both static and dynamic groups.

Since the common group key is to be agreed upon from the contributions of all the members in a sequential manner, this method is not suitable for large groups i.e., the key management operations increase exponentially with respect to the number of members in a group. So scalability is a great issue and it does not address other security services such as key integrity, entity authentication, non-repudiation and access control.
2.3.3 Tree based Group Diffie-Hellman (TGDH)

The Tree Based Group Diffie-Hellman (TGDH) contributory key agreement protocol (Blundo C et al 1993) is robust and efficient in the sense that, it can deal with network partition and the number of rounds for re-keying is limited by $O(\log_2 n)$, where $n$ is the number of members currently in the group.

![Figure 2.14: Tree based Group Diffie-Hellman key distribution tree](image)

In TGDH, all members maintain an identical virtual binary key tree which may or may not be balanced. The nodes in the key tree are denoted by $<l, v>$ where $l$ is the level in the tree and $v$ indicates the node in level $l$. The root node is labeled as $<0, 0>$ and the two children of a node $<l, v>$ are labeled as $<l+1, 2v>$ and $<l+1, 2v+1>$. Every node except the root node is associated with a secret key $K_{<l, v>}$ and a blinded key $BK_{<l, v>} = f(K_{<l, v>})$ where $f(x) = g^x \mod p$. The extended Diffie hellman protocol is used to share the secret key between each node and to compute a common key. A sponsor of a sub tree is the member hosted on the right most leaf in the sub-tree and the sponsor of a leaf node is the member hosted on the rightmost leaf node of the lowest sub-tree, where this leaf node belongs to. For every join, the
sponsor is determined for the joining member and the sponsor generates new private share and computes all secret keys and blinded keys from its leaf node to the root node (Burmuster M and Desmedt Y G 1996). Similarly the root key is computed in one round for single leave.

The computation of a common key for six members M1, M2, M3, M4, M5 and M6 using TGDH protocol is shown in the Figure 2.12. Here the publicly available components are g which is a primitive root of p. The mod operator is not shown in the illustration for simplicity. Considering the node<2, 0> which calculates the blinded key BK<2, 0> = f ((BK<3, 0>)*K<3, 1>) = f ( (BK<3, 1>)*K<3, 0>) = gga1a2. After computing the blinded key values, they are broadcasted to the group members. The member M1 calculates K<1, 0>, BK<1, 0> and it receives BK<1, 1> broadcasted from the members hosted in the right sub tree and it calculates the root key which is equal to BK<0, 0> = f (BK<1, 1>*K<1, 0>). Eventually all the other members compute the root key as M1. The problem with this protocol is seamless communication and during re-keying process, the communication has to be stopped until the new key is computed. In order to overcome these difficulties another protocol is discussed in the next section.

2.3.4 Block-Free TGDH

The Block-Free Tree Based Group Diffie-Hellman (BF-TGDH) is an extension of the TGDH protocol. The BF-TGDH protocol (Xukai Zou and Byrav Ramamurthy) uses two kinds of keys viz., front-end keys and back-end keys. The front end key can be computed by all group members whereas for back-end keys, each member can compute other member keys except one key which is intended for that particular member. Whenever a member leaves, the remaining members will switch over to the back end key which is not possessed by the leaving member and the group communication may be continued along with the re-computation of all keys in a background process which is illustrated in Figure 2.13. Here M1, M2, M3, M4, M5 and M6 represent the users and a1, a2, a3, a4, a5 and a6 are the private values of above mentioned users respectively. D1, D2, D3, D4, D5 and D6 are dummy
members, the corresponding dummy values are denoted as \(d_1, d_2, d_3, d_4, d_5\) and \(d_6\) and \(g\) is a public Diffie-Hellman component. The key calculated by \(M_1\) is described as follows.

The notation \((a_1\ldots d_i\ldots a_m)\) represents \(k(a_1\ldots d_i\ldots a_m) \rightarrow g k(a_1\ldots d_i\ldots a_m)\) where \(k(a_1\ldots d_i\ldots a_m)\) is a dummy secret key, \(BK(a_1\ldots d_i\ldots a_m) = g k(a_1\ldots d_i\ldots a_m)\) is a dummy blinded key and \(d_i\)'s \((e_i = g d_i)\) are dummy private shares (dummy disguised public shares) which are generated by an off-line shares generator. For example, let us consider \(M_1\). \(M_1\) can compute \((a_1d_2)\) (i.e., \(g a_1d_2 \rightarrow g g a_1d_2\)) and broadcast \(BK(a_1d_2) = g g a_1d_2\). Then \(M_1\) computes \((a_1d_2a_3)\) (i.e., \(g g a_1d_2a_3 \rightarrow g g g a_1d_2a_3\) after receiving \(g a_3\) from \(M_3\)) and \((a_1a_2d_3)\) from \(g d_3\) which is generated by the off-line shares generator. Finally, \(M_1\) computes \((a_1d_2a_3a_4a_5a_6)\) when receiving \(g g g a_1d_2a_3a_4a_5a_6\), \((a_1a_2d_3a_4a_5a_6)\) when receiving \(g a_1d_2a_3a_4a_5a_6\), \((a_1a_2a_3d_4a_5a_6)\) when receiving \(g g d_4a_5a_6\), \((a_1a_2a_3a_4d_5a_6)\) when receiving \(g a_4d_5a_6\), and \((a_1a_2a_3a_4a_5d_6)\) when receiving \(g a_4g d_5a_6\). It is evident that \(M_1\) cannot calculate \(d_1a_2a_3a_4a_5a_6\) and can be calculated by all the other members. So whenever \(M_1\)
leaves the group, the remaining members manage their session with the above said key until the new root key is calculated in a block free manner.

Since BF-TGDH is an extension of TGDH, it overcomes the problem of seamless communication. But this leads to the problem of storage complexity and computational efficiency. Every member has to calculate n root keys instead of one root key. Since lots of keys are used, sharing of these keys lead to the problem of increase in the size of messages. Regarding security issues, when two leaving members collude, they can compute the dummy root keys. So it loses its perfect forward secrecy.

2.4 DISTRIBUTED SUBGROUP APPROACHES

In distributed subgroup approach, the large group is split into small groups. Different controllers are used to manage each sub group, minimizing the problem of concentrating the work on a single place. Some of the popular distributed subgroup approaches such as (i) IOLUS (ii) Dual Encryption Protocol and (iii) Hybrid Rekeying Mechanism are discussed in the sections to follow.

2.4.1 IOLUS

Mitra S (1997) proposed IOLUS, a framework with a hierarchy of agents that splits the large group into small sub groups. A Group Security Agent (GSA) manages each sub group. The GSAs are also grouped into top-level group.

IOLUS uses independent keys for each sub-group, for example K1 is the key used subgroup 1 as shown in Figure 2.16, it means the changes that affect a sub group are not reflected in other sub groups. In addition, absence of a central controller contributes to the fault tolerance of the system i.e. if a sub group controller fails, only its sub group is affected. IOLUS is designed with special features like scalability, robustness and independence.
Even though it is scalable, a mechanism is needed to handle overloaded sub group agents due to formation of new sub groups under the same sub group agent. When one of the GSA and GSC fails, data exchange between groups is not possible. Mostly it affects data path. This occurs in the sense that there is a need for translating the data that goes from one sub group to another. This become even more problematic when it is taken in to account that the GSA has to manage the sub group and perform the translations needed.

### 2.4.2 Dual Encryption Protocol

Dual Encryption Protocol (DEP) supports secure one-to-many group communication (Dondeti L R et al 1999), dynamic group membership and it is scalable. It uses hierarchical sub grouping of multicast members to address scalability. Each subgroup is managed by a subgroup manager (SGM). SGMs are trusted routers or hosts in the network that can handle the workload of managing a subgroup in the multicast group.

There are three kinds of KEKs and one Data Encryption Key (DEK). KEK1 is shared between SGMi and its subgroup members.KEK2 is shared between the Group Controller (GC) and the members of subgroup i, excluding SGMi. Finally, GC shares
KEKi3 with SGMi. In order to distribute the DEK to the members, the GC generates and transmits a package containing the DEK encrypted with KEKi2 and encrypted again with KEKi3 (SGMs KEK). On receiving the package, SGMi decrypts its part of the message using KEKi3 and recovers the DEK encrypted with its subgroup KEK (KEKi2), which is not known by the SGMi. SGMi encrypts this encrypted DEK using KEKi1 shared with its subgroup members and sends it out to subgroup i. Each member of subgroup i decrypts the message using KEKi1 and then, decrypting the message using KEKi2 (shared with GC), recovers DEK. The DEK cannot be recovered for any entity that does not know both keys. Even though there are third parties involved in the management (SGMs), they do not have access to the group key (DEK). When the subgroup i’s memberships changes the SGMi changes KEKi1 and sends it to its members. Future DEK changes cannot be accessed for members of subgroup i that did not received the new KEKi1. However, evicted members that did not receive KEKi1 can still access the group communication until the DEK is changed and hence this compromises forward secrecy. Another disadvantage of this method is a trivial collusion attack exists i.e. the leaving user colludes with a participant-SGM and the participant-SGM can get the KEK.

2.5 HYBRID REKEYING MECHANISM

Ivan Y.K Pang and Henry C.B Chan (2003) combined the features of distributed and centralized key management schemes and they proposed a Hybrid Rekeying Mechanism (HRM) for supporting secure multicast of multimedia data over the Internet. This architecture combines IOLUS and omitted key approach of Centralized Flat Key Management (CFKM). The IOLUS framework seeks to directly address the problem of scalability by completely doing away with the idea of single flat secure multicast group. Instead, IOLUS substitutes the notion of a secure distribution tree. The secure distribution tree is composed of a number of smaller secure multicast subgroups arranged in a hierarchy to create a single virtual secure multicast group. The subgroups are managed by the omitted key approach. According to this approach each user in a group of n member has to store n-1 keys, i.e. keys of all other members except the key having his ID. If a member has ID 2 then he will
have keys $k_1$, $k_3$, $k_4$, $k_5$… $k_8$ in a group of 8 members. So now if this member with ID 2 leaves, the new group key will be encrypted with the “omitted key” $K_2$ so that the evicted member cannot decrypt it. But this scheme has the disadvantage that each member has to store $n-1$ keys and this increases the storage complexity of the user. Also in case of bulk leave the members will be able to collude and decrypt the new group key.

In order to reduce the storage complexity and to avoid collusion attacks, an efficient hybrid re-keying mechanism is proposed which combines IOLUS (Mittra S 1997) and One way function tree (Balenson D et al 2000) and is illustrated in Figure 2.18. This is similar to the previous hybrid method except now in the subgroups level one-way function tree is implemented. In one-way function tree, the keys are computed up the tree from the leaves to the root. This approach reduces re-keying broadcasts to about $\log n$ keys, where $n$ is the number of group members. This algorithm provides complete forward and backward secrecy. Newly admitted group members cannot read previous messages, and evicted members cannot read future messages. In addition to previous method, this algorithm has the option of being member contributory i.e., the members can be allowed to contribute their part to the group key. But in subgroup level the keys need to be computed for every join and leave.

Figure 2.17: System Architecture of hybrid (IOLUS+CFKM) rekeying mechanism
2.5.1 Comparative Analysis

The detailed comparison of the complexities of the above said methods are tabulated in Table 2.2. Here n represents the number of members in each subgroup and m represents the number of subgroups.

Table 2.2: Complexity Analysis of the Hybrid Methods

<table>
<thead>
<tr>
<th>COMPLEXITIES INVOLVED</th>
<th>(IOLUS + CFKM)</th>
<th>(IOLUS + OFT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY COMPLEXITY</td>
<td>( M(n + 1) + 2 )</td>
<td>( m(n + 1) + 2 )</td>
</tr>
<tr>
<td>STORAGE COMPLEXITY</td>
<td>( m(n^2 + 5n + 6) + 5 )</td>
<td>( (\log(n+4)n+3)m + 2 )</td>
</tr>
<tr>
<td>COMMUNICATION COMPLEXITY</td>
<td>JOIN: 3</td>
<td>JOIN: 3</td>
</tr>
<tr>
<td></td>
<td>DEPART: ( m + 2 )</td>
<td>DEPART: ( m + 3 )</td>
</tr>
<tr>
<td>COMPUTATION COMPLEXITY</td>
<td>JOIN: 3</td>
<td>LOG( n + 2 )</td>
</tr>
<tr>
<td></td>
<td>DEPART: ( m + 3 )</td>
<td>( m + \log n )</td>
</tr>
</tbody>
</table>
From the Table 2.2, it is inferred that the storage complexity is less in (IOLUS + OFT) method compared with (IOLUS + CFKM) method but the computation complexity is more in the first case. The merits and the shortcomings of existing scalable secure multicasting protocols are tabulated in Table 2.3. In this table n denotes the no of members in the multicast group, l denotes the no of subgroups in the group, Î denotes the average subgroup size, c denotes the no of children of the sender of the multicast data and d represents the degree of the hierarchical distribution tree.

Table 2.3: Comparison of various key management schemes

<table>
<thead>
<tr>
<th>Parameters</th>
<th>OFT</th>
<th>CFKM</th>
<th>IOLUS</th>
<th>DEP</th>
<th>Key Graph</th>
<th>HBT</th>
<th>Boolean minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of keys in the multicast group</td>
<td>4n-3 O(n)</td>
<td>2logn+1 O(logn)</td>
<td>n+i+1 O(n)</td>
<td>n+i+1+c O(n)</td>
<td>n(1+logn) O(n)</td>
<td>2logn+1 O(logn)</td>
<td>2logn+1 O(n)</td>
</tr>
<tr>
<td>No of keys managed by the sender</td>
<td>4n-3</td>
<td>2logn+1</td>
<td>2</td>
<td>c+2</td>
<td>(d+2) (h-1)/2</td>
<td>2logn+1</td>
<td>2logn+1</td>
</tr>
<tr>
<td>No of keys at a member</td>
<td>O(log n)</td>
<td>O(logn)</td>
<td>3</td>
<td>4</td>
<td>d/(d-1) O(logn)</td>
<td>logn+1</td>
<td></td>
</tr>
<tr>
<td>No of keys at a SGM</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1 affects n scalability</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No of messages at join</td>
<td>O(log n)</td>
<td>O(logn)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(logn)</td>
<td>Not dealt</td>
</tr>
<tr>
<td>No of messages at leave</td>
<td>O(log n)</td>
<td>O(logn)</td>
<td>O(Î)</td>
<td>O(Î)</td>
<td>O(logn)</td>
<td>O(logn)</td>
<td>&lt; logn</td>
</tr>
<tr>
<td>Total key encryptions during data</td>
<td>O (1)</td>
<td>O (1)</td>
<td>O (l)</td>
<td>O (l+c)</td>
<td>O (l)</td>
<td>O (1)</td>
<td>O (l)</td>
</tr>
<tr>
<td>transmissions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No of key encryptions at the sender</td>
<td>O (1)</td>
<td>O (1)</td>
<td>O (l)</td>
<td>O (c)</td>
<td>O (l)</td>
<td>O (1)</td>
<td>O (l)</td>
</tr>
<tr>
<td>Vulnerable to collusions</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Since almost all the encryption algorithms are available in open literature, the security level in multicasting is mainly determined by how the keys used are maintained and distributed in a secret manner. So, as stated earlier key management plays a vital role in providing security for multicasting. The next chapter explains the proposed energy efficient topology aware-key management scheme which ensures backward secrecy.