Key Bundles and Parcels:
Secure Communication in Many Groups

Eunjin Jung, Xiang-Yang Alex Liu, Mohamed G. Gouda
{ejung, alex, gouda}@cs.utexas.edu

Department of Computer Sciences
The University of Texas at Austin

Abstract. We consider a system where each user is in one or more elementary groups. In this system, arbitrary groups of users can be specified using the operations of union, intersection, and complement over the elementary groups in the system. Each elementary group in the system is provided with a security key that is known only to the users in the elementary group and to the system server. Thus, for any user u to securely multicast a data item d to every user in an arbitrary group G, u first forwards d to the system server which encrypts it using the keys of the elementary groups that comprise G before multicasting the encrypted d to every user in G. Every elementary group is also provided with a key tree to ensure that the cost of changing the key of the elementary group, when a user leaves the group, is small. We describe two methods for packing the key trees of elementary groups into key bundles and into key parcels. Packing into key bundles has the advantage of reducing the number of encryptions needed to multicast a data item to the complement of an elementary group. Packing into key parcels has the advantage of reducing the total number of keys in the system. We apply these two methods to a class of synthetic systems: each system has 10000 users and 500 elementary groups, and each user is in 2 elementary groups on average. Simulations of these systems show that our proposals to pack key trees into key bundles and key parcels live up to their promises.

1 Introduction

We consider a system that consists of n users denoted u_i, $0 \leq i < n$. The system users share one security key, called the system key. Each user u_i can use the system key to encrypt any data item before sending it to any subset of the system users, and can use it to decrypt any data item after receiving it from any other system user. (Examples of such systems are secure multicast systems [3], [7], [14], [15], secure peer-to-peer systems [12], and secure wireless networks [4].)

When a user u_i leaves the system, the system key needs to be changed so that u_i can no longer decrypt the encrypted data item exchanged within the system. This requires to add a server S to the system and to provide each system user u_j with an individual key K_j that only user u_j and server S know. When a user u_i leaves the system, server S changes the system key and sends the new key to each user u_j, other than u_i, encrypted using its individual key K_j. The cost of this rekeying scheme, measured by the number of needed encryptions, is $O(n)$, where n is the number of users in the system.

Clearly, this solution does not scale when the number of users become large. More efficient rekeying schemes have been proposed in [1], [2], [8], [9], [10], and [13]. A particular efficient rekeying scheme [14] and [15] is shown to cost merely $O(\log n)$ encryptions. This scheme is extended in [5], [6], and [16], and is shown to be optimal in [11], and has already been accepted as an Internet standard [14].

This scheme is based on a distributed data structure called a key tree. A *key tree* is a directed, incoming, rooted, balanced tree where each node represents a key. The root of the tree represents the system key and each leaf node represents the individual key of a system user. The number of leaf nodes is n, which is the number of users in the system. Each user knows all the keys on the directed path from its individual key to the root of the tree, and the server knows all the keys in the key tree. Thus, in a binary key tree, each user knows $\lceil \log_2 n \rceil + 1$ keys, and the server knows $(2n-1)$ keys.

An example of a key tree for a system of 8 users is depicted in Figure 1(a). The root of the key tree represents the system key $K_{01234567}$ that is known to all users in the system. Each user also knows all the keys on the directed path from its individual key to the root of the key tree. For example, user u_7 knows all the keys K_7, K_{67}, K_{4567}, and $K_{01234567}$.

Figure 1(a) and 1(b) illustrates the protocol for updating the system key when user u_7 leaves the system. In this case, the system server S is required to change the keys $K_{01234567}$, K_{4567}, and K_{67} that user u_7 knows. To update these keys, S selects new keys $K_{0123456(7)}$, $K_{456(7)}$, and $K_{6(7)}$, encrypts them, and sends them to the users that need to know them. To ensure that u_7 cannot get a copy of the new keys, S needs to encrypt the new keys using

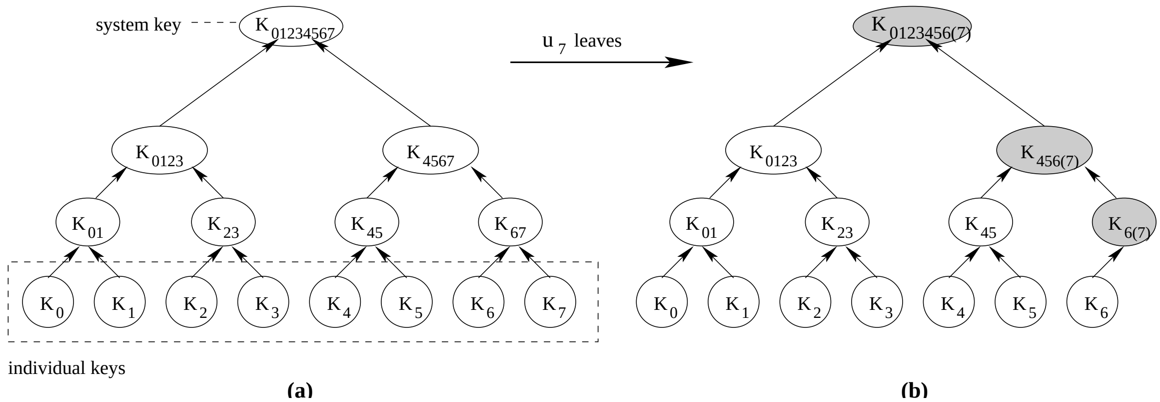


Fig. 1. A binary key tree before and after u_7 leaves

keys that u_7 does not know. Therefore, S encrypts the new $K_{0123456(7)}$ with the old K_{0123}, encrypts the new $K_{0123456(7)}$ and the new $K_{456(7)}$ with the old K_{45}, encrypts the new $K_{0123456(7)}$, the new $K_{456(7)}$, and the new $K_{6(7)}$ with K_6. Then, S multicasts the encrypted keys to the corresponding holders of these keys. The protocol can be specified as follows.

$$S \rightarrow u_0, \cdots, u_6: \ \{u_0, u_1, u_2, u_3\}, \ K_{0123} < K_{0123456(7)} | \text{chk} >$$
$$S \rightarrow u_0, \cdots, u_6: \ \{u_4, u_5\}, \ K_{45} < K_{0123456(7)} | K_{456(7)} | \text{chk} >$$
$$S \rightarrow u_0, \cdots, u_6: \ \{u_6\}, \ K_6 < K_{0123456(7)} | K_{456(7)} | K_{6(7)} | \text{chk} >$$

This protocol consists of three steps. In each step, server S broadcasts a message consisting of two fields to every user in the system. The first field defines the set of the intended ultimate destinations of the message. The second field is an encryption, using an old key, of the concatenation of the new key(s) and a checksum computed over the new key(s). Note that although the broadcast message is sent to every user in the system, only users in the specified destination set have the key used in encrypting the message and so only they can decrypt the message.

The above system architecture is based on the assumption that the system users constitute a single group. In this paper, we extend this architecture to the case where the system users form many groups.

2 Groups and Group Algebra

Assume that the system has m, $m \geq 1$, elementary groups: each elementary group is a distinct subset of the system users and one elementary group has all the system users. Every elementary group has a unique identifier G_j, $0 \leq j \leq m-1$. The identifier for the elementary group that has all users is G_0. As an example, Figure 2 illustrates a system that has eight users u_0 through u_7 and five elementary groups G_0, G_1, G_2, G_3, and G_4.

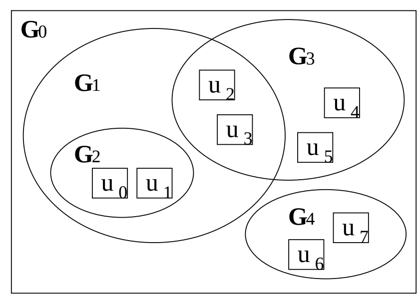


Fig. 2. A sample system

The system needs to be designed such that any user u_i can securely multicast data items to all users in any elementary group G_j. Moreover, any user u_i can securely multicast data items to all users in any group, where a group is defined recursively according to the following four rules:

i. Any of the elementary groups $G_0, \cdots, G_{m-1}$ is a group.
ii. The union of any two groups is a group.
iii. The intersection of any two groups is a group.
iv. The complement of any group is a group.(Note that the complement of any group G is the set of all users in G_0 that are not in G)

Thus, the set of groups is closed under the three operations of union, intersection, and complement.

Each group can be defined by a group formula that includes the following symbols.

– G_0 through G_{m-1}
– $\vee$ for union

– $\wedge$ for intersection
– $\neg$ for complement

Group formulae can be manipulated using the well-known laws of algebra: associativity, commutativity, distribution, De-Morgan's, and so on. For example, the group formula

$$G_1 \vee \neg(\neg G_2 \wedge G_1)$$

can be manipulated as follows:

$$\begin{aligned}
&G_1 \vee \neg(\neg G_2 \wedge G_1)\\
&\quad = \{\text{by De Morgan's}\}\ G_1 \vee (\neg\neg G_2 \vee \neg G_1)\\
&\quad = \{\text{by associativity of } \vee\}\ G_1 \vee \neg\neg G_2 \vee \neg G_1\\
&\quad = \{\text{by definition of complement}\}\ G_1 \vee G_2 \vee \neg G_1\\
&\quad = \{\text{by commutativity of } \vee\}\ G_1 \vee \neg G_1 \vee G_2\\
&\quad = \{\text{by definition of complement}\}\ G_0 \vee G_2\\
&\quad = \{\text{by definition of } \vee\}\ G_0
\end{aligned}$$

From this formula manipulation, it follows that the group defined by the formula $G_1 \vee \neg(\neg G_2 \wedge G_1)$ is the set of all system users. Thus, for a user u_i to securely multicast a data item d to every user in the group $G_1 \vee \neg(\neg G_2 \wedge G_1)$, it is sufficient for u_i to securely broadcast d to every user in the system.

In the rest of this paper, we consider solutions for the following problem. How to design the system so that any system user u_i can securely multicast data items to any group G in the system. Any reasonable solution for this problem needs to take into account that the users can leave any elementary group in the system or leave the system altogether, and these activities may require to change the security keys associated with the elementary groups from which users leave. In particular, the solution should utilize key trees, discussed in Section 1, that can reduce the cost of changing the security keys from $O(n)$ to $O(\log n)$, where n is the total number of users in the system.

The above problem has many applications. As a first example, consider a peer-to-peer music file sharing system that has four elementary groups: Rock, Jazz, Blues, and Do-Not-Disturb. A user u_i in this system may wish to securely distribute a song of Louis Armstrong to all interested users. In this case, user u_i securely multicasts the song to all users in the group, Jazz$\wedge\neg$Do-Not-Disturb.

As a second example, consider a student registration system in some university. This system has m elementary groups G_0 through G_{m-1}, where each G_i is a list of the students registered in one course section. A professor who is teaching three sections G_5, G_6, G_7 of the same course, may wish to securely multicast any information related to the course to all the students in the group $G_5 \vee G_6 \vee G_7$.

3 Key Bundles

The above problem suggests the following simple solution (which we show below that it is ineffective). First, assign to each elementary group G_j a security key to be shared by all the users of G_j. Second, assign to the complement $\neg G_j$ of each elementary group G_j a security key to be shared by every member of this complement. Third, provide a key tree for each elementary group and another key tree for its complement. Note that the two key trees provided for an elementary group and its complement span all the users in the system. Thus, these two trees can be combined into one *complete* key tree that spans all system users in the system. Figure 3 shows the four complete key trees that are provided for the four elementary groups and their complements in the system in Figure 2.

From Figure 3(a), the key for the elementary group G_1 is K_{0123} and the key for its complement is K_{4567}. From Figure 3(b), the key for the elementary group G_2 is K_{01} and the key for its complement is K_{234567}. From Figure 3(c), the key for the elementary group G_3 is K_{2345}, and the key fro its complement is K_{0167}. From Figure 3(d), the key for the elementary group G_4 is K_{67}, and the key for its complement is K_{012345}.

Note that these complete trees have the same key for group G_0, and the same individual key for each user. Nevertheless, the total number of distinct keys in these complete trees is 19, which is relatively large for this rather simple system. In general, this method requires $O(mn)$ keys, where m is the number of elementary groups and n is the number of users in the system.

To reduce the total number of needed keys, several elementary groups can be added to the same complete key tree, provided that these elementary groups are "nonconflicting". This idea suggests the following three definitions of nonconflicting elementary groups, bundles, and bundle covers.

Two elementary groups are *nonconflicting* if and only if either their intersection is empty or one of them is a subset of the other. In the system example in Figure 2, the three elementary groups G_0, G_1 and G_2 are nonconflicting since G_1 is a subset of G_0, and G_2 is a subset of G_1. On the other hand, the two elementary groups G_1 and G_3 are conflicting, because they share two users u_2 and u_3 and neither group is a subset of the other.

A *bundle* of a system is a maximal set of nonconflicting elementary groups of the system. In the system example in Figure 2, the four elementary groups G_0, G_1, G_2, G_4 constitute one bundle B_0, and the four elementary groups G_0, G_2, G_3, G_4 constitute a second bundle B_1.

A *bundle cover* of a system is a set $\{B_0, \cdots, B_{m-1}\}$ of system bundles such that the following two conditions hold:

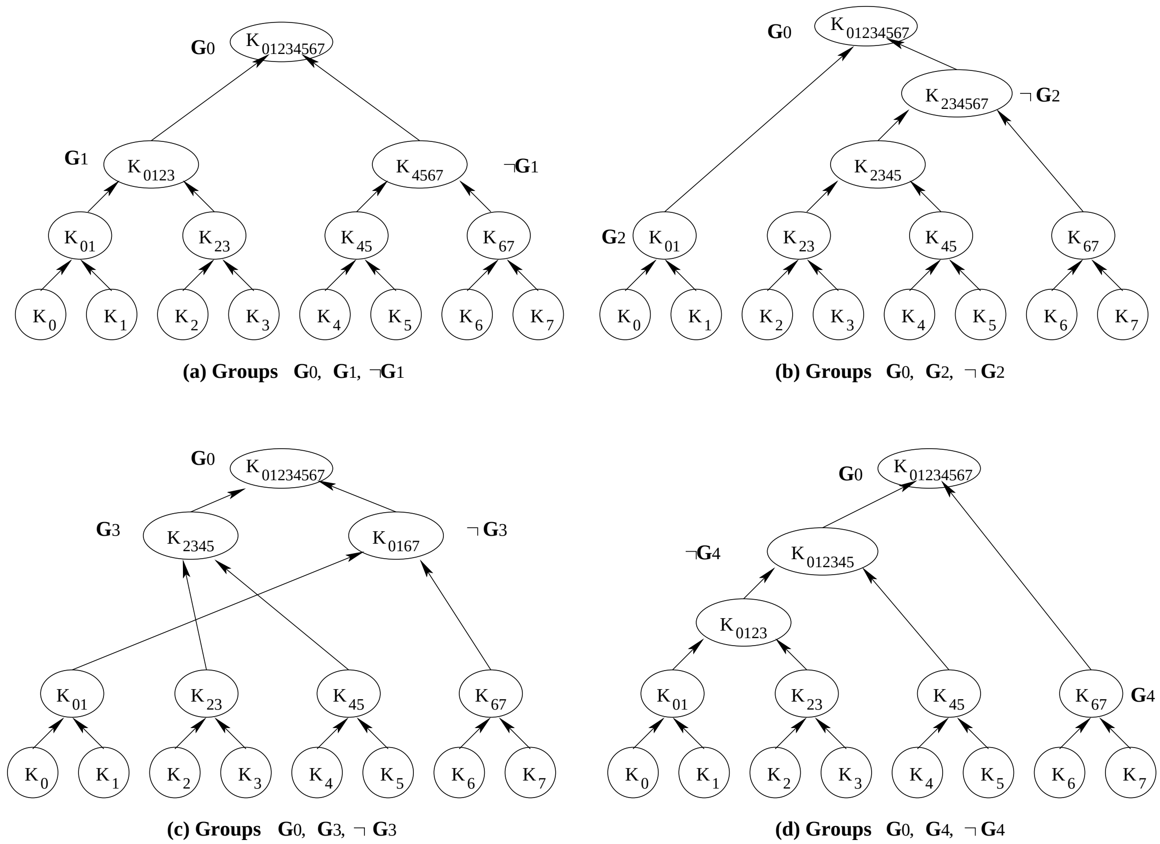


Fig. 3. The complete key trees for the elementary groups and their complements

i. *Completeness*: Each elementary group of the system appears in some bundle B_i in the bundle cover.
ii. *Compactness*: Each bundle B_i has at least one elementary group that does not appear in any other bundle B_j in the bundle cover.

Note that the set $\{B_0, B_1\}$, where $B_0 = \{G_0, G_1, G_2, G_4\}$ and $B_1 = \{G_0, G_2, G_3, G_4\}$, is a bundle cover for the system in Figure 2.

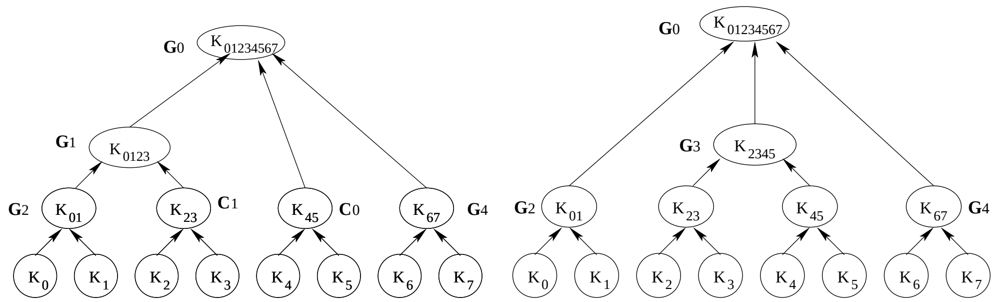
The security keys for the elementary groups in a bundle can be arranged in a complete key tree. For example, Figure 4(a) shows the complete key tree for B_0. In this tree, the key for group G_0 is $K_{01234567}$, the key for group G_1 is K_{0123}, the key for G_2 is K_{01}, and the key for G_4 is K_{67}. Note that users u_4 and u_5 in G_0 do not belong to any other elementary group in the bundle, and so they are viewed as forming a complement group C_0 whose key is K_{45}. We refer to a complete key tree that corresponds to a bundle as a *key bundle*.

Figure 4(b) shows the complete key bundle for B_1. Note that in this bundle every user in G_0 is also in another elementary group. Thus, the resulting complete key tree does not have a complement group as in the former key tree in Figure 4(a).

Comparing the two key bundles in Figures 4(a) and 4(b), one observes that each of the elementary groups G_0, G_2, and G_4 appear in both key bundles because none of them conflict with any elementary group or any group in the system. One also observes that each of these groups has the same group key in both key bundles, and that the individual key of each user is the same in both key bundles. Note that these key bundles have only 15 distinct keys compared with the 19 distinct keys in the four complete trees in Figure 3. This represents more than 20% reduction in the total number of keys in the system.

The system server S knows the two key bundles in Figure 4, and each user u_i knows only the keys that exist on the paths from its individual key K_i to the key of group G_0. Thus, each user u_i needs to collaborate with the system server S in order to securely multicast data items to any elementary group or any group that can be defined by intersection, union, and complement of elementary groups. This point is illustrated by the following four examples.

For the first example, assume that user u_0 wants to securely multicast a data item d to every user in group G_4. In this case, user u_0 can execute the following protocol.



(a) $B_0 = \{G_0, G_1, G_2, G_4\}$ (b) $B_1 = \{G_0, G_2, G_3, G_4\}$

Fig. 4. The complete key trees for the two bundles B_0 and B_1

$$
\begin{aligned}
u_0 \to S &: K_0 < d|G_4|\text{chk} > \\
S \to u_0, \cdots, u_7 &: G_4, \quad K_{67} < d|u_0|\text{chk} >
\end{aligned}
$$

This protocol consists of two steps. In the first step, user u_0 sends a message $K_0 < d|G_4|\text{chk} >$ to server S. This message consists of three concatenated fields, namely the data item d, its intended destination G_4, and the checksum chk; the message is encrypted by the individual key K_0 of user u_0. In the second step, server S multicasts the message G_4, $K_{67} < d|u_0|\text{chk} >$ where the second field consists of the data item d, the message source u_0, and the checksum chk and is encrypted with the group key of G_4.

For the second example, assume user u_1 wants to securely multicast a data item d to the users in either group G_1 or G_3, namely the users in the union of G_1 and G_3. In this case, user u_1 can execute the following protocol.

$$
\begin{aligned}
u_1 \to S &: K_1 < d|G_1 \vee G_3|\text{chk} > \\
S \to u_0, \cdots, u_7 &: G_1 \vee G_3, \quad K_{0123} < d|u_1|\text{chk} >, \\
& K_{2345} < d|u_1|\text{chk} >
\end{aligned}
$$

In the second step of this protocol, server S multicasts the message $G_1 \vee G_3$, $K_{0123} < d|u_1|\text{chk} >$, $K_{2345} < d|u_1|\text{chk} >$ to the two groups G_1 and G_3. The users in group G_1 can get d by using the group key K_{0123} to decrypt $K_{0123} < d|u_1|\text{chk} >$ and the users in group G_3 can get d by using the group key K_{2345} to decrypt $K_{2345} < d|u_1|\text{chk} >$. Note that if it is u_2 who wants to send d to $G_1 \vee G_3$, then since u_2 belongs to both G_1 and G_3, u_2 already knows both K_{0123} and K_{2345}. Therefore, u_2 can send the encrypted d directly to the users in G_1 and G_3 as follows:

$$
\begin{aligned}
u_2 \to u_0, \cdots, u_7 &: G_1 \vee G_3, \quad K_{0123} < d|u_2|\text{chk} >, \\
& K_{2345} < d|u_2|\text{chk} >
\end{aligned}
$$

For the third example, assume that user u_4 wants to send a data item d to all the users in the intersection of G_1 and G_3. In this case, user u_4 can execute the following protocol.

$$
\begin{aligned}
u_4 \to S &: K_4 < d|G_1 \wedge G_3|\text{chk} > \\
S \to u_0, \cdots, u_7 &: G_1 \wedge G_3, \quad K_{0123} < K_{2345} < d|u_4|\text{chk} >>
\end{aligned}
$$

In the second step of this protocol, server S multicasts a message $G_1 \wedge G_3$, $K_{0123} < K_{2345} < d|u_4|\text{chk} >>$ to the group $G_1 \wedge G_3$. Here the concatenation of d, u_4 and chk is encrypted by both the group key of G_1, which is K_{0123}, and the group key of G_3, which is K_{2345}. The encrypted message can only be decrypted by the users that are in both G_1 and G_3 because only these users know the two group keys K_{0123} and K_{2345}.

For the fourth example, assume that user u_5 wants to send a data item d to all the users in the complement of group G_1. In this case, user u_5 executes the following protocol.

$$
\begin{aligned}
u_5 \to S &: K_5 < d|\neg G_1|\text{chk} > \\
S \to u_0, \cdots, u_7 &: C_0 \vee G_4, \quad K_{45} < d|u_5|\text{chk} >, \\
& K_{67} < d|u_5|\text{chk} >
\end{aligned}
$$

After server S receives this message, it translates $\neg G_1$ to $C_0 \vee G_4$ then multicasts the message $G_c \vee G_4, K_{45} < d|u_5|\text{chk} >, K_{67} < d|u_5|\text{chk} >$. The users in group G_c can get d using the group key K_{45}, and the users in group G_4 can get d using the group key K_{67}.

4 Construction of Key Bundles

In this section, we describe a procedure that can be used by the server of a system to construct and maintain key bundles for that system. This procedure consists of two algorithms. The first algorithm, presented in Section 5.1, constructs a bundle cover for the given system. The second algorithm, presented in Section 5.2, computes a complete key tree (i.e. a key bundle) for each bundle in the bundle cover constructed by the first algorithm.

4.1 Algorithm for Bundle Construction

This algorithm takes any given system with m elementary groups $G_0, \cdots, G_{m-1}$ and constructs a bundle cover $\{B_0, \cdots, B_{r-1}\}$ for the given system. Note that $r \leq m$.

In this algorithm, the given system is represented by a $m \times m$ boolean matrix C, where each element $C[i][j]$ is defined as follows:

$$C[i][j] = \begin{cases} \textbf{false} & \text{if } G_i \cap G_j = \phi \text{ or } G_i \subset G_j \text{ or } G_j \subset G_i \\ \textbf{true} & \text{otherwise} \end{cases}$$

The algorithm starts with m empty bundles, $B_0, \cdots, B_{m-1}$. Then the algorithm proceeds to add elementary groups of the given system to the bundles, one by one, until each elementary group is added to at least one bundle. Finally, the algorithm keeps the bundles $B_0, \cdots, B_{r-1}$ that have elementary groups and discards the remaining empty bundles $B_r, \cdots, B_{m-1}$.

The algorithm uses an array done$[0..m-1]$ of m boolean elements to keep track of the elementary groups that have already been added to bundles. Initially, every done$[j]$ has the value **false**, and when the elementary group G_j is added to some bundle, then done$[j]$ is assigned the value **true**.

The bundle construction algorithm is specified in Figure 5(a). Note that Lines 7 and 12 is this algorithm call a boolean function NOCONFLICT(B_r, G_y). This function is specified in Figure 5(b).

As an example, if this algorithm is applied to the system in Figure 2, the algorithm constructs the bundle cover $\{B_0, B_1\}$, where

$$\begin{aligned} B_0 &= \{G_0, G_1, G_2, G_4\} \\ B_1 &= \{G_0, G_2, G_3, G_4\} \end{aligned}$$

4.2 Algorithm for Key Bundle Construction

Next we describe an algorithm that takes as input any bundle B in the bundle cover, constructed by the above algorithm, and computes a complete key tree for B. The following definition of a child is needed in stating our algorithm.

Let B be a bundle and let G and G'' be two distinct elementary groups in B. Then, G is a *child* of G'' iff $G \subset G''$ and B has no elementary group G' such that $G \subset G' \subset G''$.

```
1:   r := 0;
2:   for x = 0 to m-1
3:       if done[x] then break;
4:       B_r := B_r ∪ {G_x};
5:       done[x] := true;

6:       for y = (x+1) to m-1
7:           if ¬done[y] and NOCONFLICT(B_r, G_y)
8:           then B_r := B_r ∪ {G_y}
9:               done[y] := true
10:      endfor

11:      for y = 0 to (m-1)
12:          if NOCONFLICT(B_r, G_y)
13:          then B_r := B_r ∪ {G_y}
14:      endfor
15:      r := r+1;
16:  endfor
17:  discard the empty bundles B_r, ··· , B_{m-1}
```

(a)

```
Function NOCONFLICT(B_r, G_y):boolean
    var flag: boolean

    flag := true;
    for every G_x in B_r
        if C[x][y] then
            flag := false;
            break
    endfor
    return flag
```

(b)

Fig. 5. Bundle cover construction algorithm

The algorithm for constructing a complete key tree T for a given bundle B consists of the following five steps.

i. For every elementary group G in B, add to T a node labeled with a key K_G for group G.
ii. For every two elementary groups G and G'' in B, if G is a child of G'', then add to T a directed edge from the node labeled K_G to the node labeled $K_{G''}$.
iii. For every elementary group G in B, if G has at least one child and has one or more users that are not in any child of G, then form a complement group C that has all users in G that are not in any child of G. Also, add to T a node labeled with K_C and a directed edge from node K_C to node K_G.
iv. To every node K_G in T that does not have any incoming edges, add a binary subtree which has K_G as its

root and whose leaves are labeled with the individual keys of the users in the elementary group G.

v. To every node K_C in T, add a binary subtree which has K_C as its root and whose leaves are labeled with the individual keys of the users in the complement group C.

As an example, if this algorithm is applied to the system in Figure 2, the algorithm constructs key trees shown in Figure 4.

5 Key Parcels

A bundle is defined as a maximal set of nonconflicting elementary groups in the system. From this definition the elementary group G_0 is in every bundle since it does not conflict with any other elementary group in the system. Thus, every key bundle is a complete key tree.

This feature of bundle maximality has one advantage and one disadvantage. The advantage is that the complement of any elementary group in a bundle B_j can be expressed as the union of some other elementary groups in B_j. Thus, securely multicasting a data item to the complement of any elementary group can be carried out efficiently. The disadvantage is that the number of keys needed in each key bundle is relatively large, and so the total number of keys in the system is relatively large.

Clearly, the disadvantage of bundle maximality outweighs its advantage in systems where users never need to securely multicast data items to the complements of elementary groups. Therefore, in these systems, we use "parcels", which are not maximal, instead of bundles, which are maximal. The definitions of parcels and parcel covers are given next.

A *parcel* of a system is a set of nonconflicting elementary groups of the system.

A *parcel cover* of a system is a sequence of parcels $(P_0, \cdots, P_{s-1})$ such that the following two conditions hold:

i. *Completeness*: Each elementary group of the system appears in some parcel P_i in the parcel cover.

ii. *Compactness*: Each elementary group in each parcel P_i conflicts with at least one elementary group in each of the preceding parcels $P_0, \cdots, P_{i-1}$ in the parcel cover.

As an example, a parcel cover for the system in Figure 2 is (P_0, P_1), where $P_0 = \{G_0, G_1, G_2, G_4\}$ and $P_1 = \{G_3\}$. Figure 6 is a parcel cover (P_0, P_1) for the system in Figure 2.

The security keys for the elementary groups in a parcel can be arranged in a key tree that is not necessarily a complete tree. Figure 6(a) shows the key tree for parcel P_0 consisting of the elementary groups G_0, G_1, G_2, and G_4. Figure 6(b) shows the key tree for parcel P_1 consisting of the elementary group G_3. Note that the key tree for parcel P_1 is not a complete tree. We refer to a key tree that corresponds to a parcel as a *key parcel*.

6 Construction of Key Parcels

In this section, we describe a procedure that can be used by the server of a system to construct and maintain key parcels for that system. This procedure consists of two algorithms.

The first algorithm constructs a parcel cover for any given system. This algorithm takes any given system with m elementary groups $G_0, \cdots, G_{m-1}$ and constructs a parcel cover $(P_0, \cdots, P_{s-1})$ for the given system, $s \leq m$. This parcel cover construction algorithm uses the same data structures and the same NOCONFLICT function used in the bundle cover construction algorithm in Figure 5(b). The parcel cover construction algorithm is shown in Figure 7.

```
s := 0;
for x = 0 to m-1
    if done[x] then break;
    P_s := P_s ∪ {G_x};
    done[x] := true;

    for y = (x+1) to k-1
        if ¬done[y] and NOCONFLICT(P_s, G_y)
        then P_s := P_s ∪ {G_y}
            done[y] := true
    endfor

    s := s+1;
endfor
discard the empty parcels P_s, ··· , P_{m-1}
```

Fig. 7. Parcel cover construction algorithm

The second algorithm computes a key tree (i.e. a key parcel) for each parcel in the parcel cover constructed by the first algorithm. This algorithm is exactly the same as the one presented in Section 5.2.

7 Simulation Results

In this section, we present the results of simulations that we carried out to demonstrate the feasibility of key bundles and key parcels. In our simulation, we used a class of synthetic systems with the following properties: