# An Efficient Key Management Scheme for Secure Data Access Control in Wireless Broadcast Services

**K.V.Rajesh, and P.Harini,**

St.ann's College of Engg. & Tech., Chirala

**Abstract**

Wireless broadcast is an effective approach to dis- seminate data to a number of users. To provide secure access to data in wireless broadcast services, symmetric key - based encryption is used to ensure that only users who own the valid keys can decrypt the data. Regarding various subscriptions, an efficient key management to distribute and change keys is in great demand for access control in broadcast services. In this paper, we propose an efficient key management scheme (namely KTR) to handle key distribution with regarding to complex subscription options and user activities. KTR has the following advantages. First, it supports all subscription activities in wireless broadcast services. Second, in KTR, a user only needs to hold one set of keys for all subscribed programs, instead of separate sets of keys for each program. Third, KTR identifies the minimum set of keys that must be changed to ensure broadcast security and minimize the rekey cost. Our simulations show that KTR can save about 45% of communication overhead in the broadcast channel and about 50% of decryption cost for each user, compared with logical key hierarchy based approaches.

***Index Terms***

*Wireless broadcast, key management, access control, key hierarchy, secure group communication, key dis-tribution*

## I. INTRODUCTION

With the ever growing popularity of smart mobile devices along with the rapid advent of wireless technology, there has been an increasing interest in wireless data services among both industrial and academic communities in recent years. Among various approaches, broadcast allows a very efficient usage of the scarce wireless bandwidth, because it allows simultaneous access by an arbitrary number of mobile clients [1]. Wireless data broadcast services have been available as commercial products for many years. In particular, the announcement of the MSN Direct Service (direct.msn.com) has further highlighted the industrial interest in and feasibility of utilizing broadcast for wireless data services.

A wireless data broadcast system consists of three compo-nents as depicted in Figure 1: (1) the broadcast server; (2) the mobile devices; and (3) the communication mechanism. The server broadcasts data on air. A user's mobile device receives the broadcast information, and filters the subscribed data according to user's queries and privileges. The specialty of the broadcast system is that (a) the server determines the schedule to broadcast all data on air, and (b) users' mobile devices listen to the broadcast channel but only retrieve data (filter data out) based on users' queries. The communication mechanism includes wireless broadcast channels and (optional) uplink channels. Broadcast channel is the main mechanism for data broadcast service providers need to ensure backward and forward secrecy [2], [3] with respect to membership dynamics. In the wireless broadcast environment, any user can monitor the broadcast channel and record the broadcast data. If the data is not encrypted, the content is open to the public and anyone can access the data. In addition, a user may only subscribe to a few programs. If data in other programs are not encrypted, the user can obtain data beyond his subscription privilege. Hence, access control should be enforced via encrypting data in a proper way so that only subscribing users can access the broadcast data, and subscribing users can only access the data to which they subscribe.

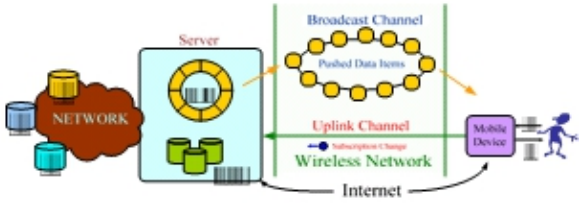


Fig. 1. A wireless data broadcast system

Symmetric-key-based encryption is a natural choice for ensuring secure data dissemination and access. The broadcast data can be encrypted so that only those users who own valid keys can decrypt them. Thus, the decryption keys can be used as an effective means for access control in wireless
data broadcast services. For example, each program has one unique key to encrypt the data items. The key is issued to the user who is authorized to receive and decrypt the data items. If a user subscribes to multiple programs, it needs an encryption key for each program. Since a user only has keys for his subscription, he cannot decrypt broadcast data and rekey messages designated to other users. At the same time, a data item can be decrypted by an arbitrary number of users who subscribe to it. This allows many users to receive the data at the same time and addresses the scalability problem, or request lost or missed keys.

Nevertheless, when a user subscribes/unsubscribes to a program, the encryption key needs to be changed to ensure that the user can only access the data in his subscription period. Consequently, a critical issue remains, i.e. how can we efficiently manage keys when a user joins/leaves/changes the service without compromising security and interrupting the operations of other users? Regarding unique features of broadcast services, we are interested in new key management schemes that can simultaneously provide security, efficiency and flexibility. A broadcast service generally provides many programs; at the same time, users may like to subscribe to an arbitrary set of programs. We envision that a user should be able to flexibly subscribe/unsubscribe to any program of interests and make changes to his subscription at any time. Hence, in addition to security and efficiency, flexibility that a user can customize his subscription at anytime is an indispensable feature of key management in broadcast services to support user subscriptions.

Two categories of key management schemes in the literature may be applied in broadcast services: (1) logic key hierarchy (LKH) based techniques [2]-[9] proposed for multicast ser-vices ; and (2) broadcast encryption techniques [10]-[16] in current broadcast services (such as satellite TV). We notice that current broadcast encryption techniques, including BISS [17], Digicipher [18], Irdeto [19], Nagravision [20], Viaccess [21], and VideoGuard [22], cannot in fact support flexibility. They normally require users to possess decryption boxes to receive the subscribed programs, and the broadcast services can only provide to users a few packages, each of which includes a fixed set of programs (TV channels). Users cannot select individual programs within a package. If a user wants to change his subscription, the user needs to request another decryption box that can decrypt the subscribed programs. Hence, in this paper, we will focus on adapting more flexible LKH-based techniques.

Nevertheless, directly applying LKH in broadcast services is not the most efficient approach. In broadcast services, a program is equivalent to a multicast group, and users who subscribe to one program form a group. Intuitively, we could manage a separate set of keys for each program, and ask a user to hold m sets of keys for his subscribed m programs. This straightforward approach is inefficient for users subscribing to many programs. If users could use the same set of keys for multiple programs, there would be fewer requirements for users to handle keys. Furthermore, when a user changes subscription, we argue that it is unnecessary to change keys for the programs to which the user is still subscribing, as long as security can be ensured. In this way, rekey cost can be reduced and fewer users will be affected. Therefore, we propose a new key management scheme, namely key tree reuse (KTR), based on two important observations: (1) users who subscribe to multiple programs can be captured by a shared key tree, and (2) old keys can be reused to save rekey cost without compromising security. KTR has two components: shared key tree and shared key management, and its contribution includes the following aspects.

Contributions. First, the proposed scheme takes advantage of a fact in broadcast services: many users subscribe to multiple programs simultaneously. In other words, programs overlap with each other in terms of users. Because existing approaches manage keys by separating programs, they turn to be demanding for the users who subscribe to many programs. Hence, this study contributes to the literature a new scheme (namely KTR) to better support subscriptions of multiple programs by exploiting the overlapping among programs. KTR let multiple programs share the same set of keys for the users who subscribe to these programs. KTR thus inherently enables users to handle fewer keys and reduces the demands of storage and processing power on resource-limited mobile devices.

Second, since multiple programs are allowed to share the same set of keys, a critical issue is how to manage shared keys efficiently and securely. We find that when keys need to be distributed to a user, it is unnecessary to change all of them. In many circumstances, when a user subscribes to new programs or unsubscribes to some programs, a large portion of keys that the user will hold in his new subscription can be reused without compromising security. KTR is a novel approach for determining which keys need to be changed and for finding the minimum number of keys that must be changed. Hence, KTR efficiently handles the rekey of the shared keys and minimizes the rekey costs associated with possible subscriptions. Our simulations show that critical keys can be employed in logical key hierarchy schemes [2], [5] to improve their performance.

The rest of the paper is organized as follows. In Section II, we present related works on group key management. In Section III, the first component of KTR is described that fully utilizes the service structure to reduce the number of keys. In Section IV, the second component of KTR is presented to reduce rekey cost when updating and distributing shared keys. In Section V, we present the results of simulations to illustrate the performance improvements in KTR. Finally, we conclude in Section VI.

## II. RELATED WORKS ON KEY MANAGEMENT

### A. Logical Key Hierarchy

Secure key management for wireless broadcast is closely related to secure group key management in networking [4]. Logical key hierarchy (LKH) is

proposed in [2], [5] that uses a key tree (depicted in Figure 2) for each group of users who subscribe the same program. The root (top node) of the tree is the data encryption key (DEK) of the program. Each leaf (bottom node) in the tree represents an individual key (IDK) of a user that is only shared between the system and the user. Other keys in the tree, namely key distribution keys (KDKs),


Fig. 2. Logical key hierarchy

are used to encrypt new DEKs and KDKs. A user only knows the keys along the path from the leaf of the user to the root of the key tree.

When a user joins or leaves the group, the server needs to change and broadcast the corresponding new keys, and this operation is called rekey, and the broadcast message of new keys is called rekey message. In our system, data and rekey messages are broadcast in the same broadcast channel to the users. Assume user u1 leaves the group (in Figure 2). The server needs to change k4, k2 and k1 so that u1 will no longer decrypt any data for this group, which is encrypted by k1. The rekey message is

$\{k^4\}_k$, $\{k^2\}_{k'}$ , $\{k^2\}_k$ , $\{k^1\}_{k'}$ , $\{k^1\}_k$

u2 4 5 2 3

where $k^i$ is the new key of $k_i$ and $\{k^i\}_k j$ means $k^i$ is encrypted by $k_j$ . When u2 receives this message, $u_2$ first decrypts $\{k^4\}_{k_{u2}}$ based on his individual key $k_{u2}$ to obtain $k^4$, then uses $k^4$ to decrypt $\{k^2\}_{k'}$ and so on to obtain $k^2$ and $k^1$.4 Similarly, other users can obtain the new keys in their own paths. It is obvious $u_1$ cannot obtain any new keys from this message, and thus the broadcast data in the future will not be decrypted by $u_1$.Now assume $u_1$ joins the group, and the server needs to change $k_4$, $k_2$ and $k_1$ so that $u_1$ cannot use the old keys to decrypt old broadcast data. Note that $u_1$ may have already eavesdropped on some broadcast data before he joined the group. If the server gives $u_1$ the old keys, $u_1$ can decrypt the eavesdropped broadcast data. The rekey message is

keys need be changed in stead of how keys are generated. [6] proposes a combination of key tree and Diffie-Hellman key exchange to provide a simple and fault-tolerant key agre agreement for collaborative groups. [23] reduces the number of rekey messages, while [9], [25] improve the reliability of rekey management. Balanced and unbalanced key trees are discussed in [5] and [26]. Periodic group re-keying is studied in [7], [8] to reduce the rekey cost for groups with frequent joins and leaves. Issues on how to maintain a key tree and how to efficiently place encrypted keys in multicast rekey packets are studied in [8], [26]. Moreover, the performance of LKH is thoroughly studied [3], [8].

## B. Broadcast Encryption

There are some other key management schemes in the literature for multicast and broadcast services. [10] used arbitrarily revealed key sequences to do scalable multicast key management without any overhead on joins/leaves. [11] proposed two schemes that insert an index head into packets for decryption. However, both of them require pre-planned subscription, which contradicts the fact that in pervasive com-puting and air data access a user may change subscriptions at any moment. In addition, [11] only supports a limited combi-nation of programs. [13] proposed a scheme to yield maximal resilience against arbitrary coalitions of non-privileged users. However, the size (entropy) of its broadcast key message is large, at least O(n) [12]. Zero-message scheme [14], [15] does not require the broadcast server to disseminate any message in order to generate a common key. But it is only resilient against coalitions of k non-privileged users, and requires every user to store $O(k\log_2(k)\log_2(n))$ keys. Naor *et al.* [16] proposed a stateless scheme to facilitate group members to obtain up-to-date session keys even if they miss some previous key distribution messages. Although this scheme is more efficient than LKH in rekey operations, it mainly handles revocation when a user stops subscription. It does not efficiently support joins, which are crucial in our system. Finally, [24], [27] pro-posed self-healing approaches for group members to recover the session keys by combining information from previous key distribution information.

$\{k^4\}_{k_u}$, $\{k^2\}_{k_u}$, $\{k^1\}_{k_u}$, $\{k^4\}_{k_4}$, $\{k^2\}_{k_2}$, $\{k^1\}_{k_1}$

The first three components are for $u_1$ to use his individual key to decrypt the new keys, and the last three are for all existing users to use their old keys to decrypt the new keys. In this way, $u_1$ will not obtain any old key.

LKH is an efficient and secure key management for multi-cast services in that each user only needs to hold $O(\log_2(n))$ keys for the user's group, and the size of a rekey message is also $O(\log_2(n))$, where $n$ is the number of group users. It is also a flexible key management approach that allows a user to join and leave the multicast group at any time. Many variations of LKH have been proposed. Because LKH simply uses independent keys, researchers developed several other approaches [23], [24] that generate new keys by exploiting the relation between child and parent keys or the relation between old and new keys. Our scheme is complementary to these schemes, since our scheme mainly examines whether Compared with LKH-based approaches, key management schemes in broadcast encryption are less flexible regarding possible subscriptions. Conforming to the current practice described in RFC2627 [2], we select binary trees to present our scheme. Note that our scheme does not require binary trees and can be applied in trees of other degrees.

## III. SHARED KEY STRUCTURE

Directly applying LKH is not efficient in broadcast services. We use a shared key structure to address the key management. In the following, we describe how a shared key structure is applied and then raise the security and efficiency problems of this scheme. We then present a novel shared key management in Section IV that ensures security and minimizes rekey cost, and also address major issues when applying KTR in a broadcast server.

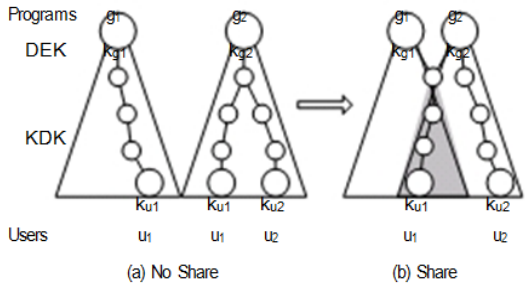


Fig. 3. Shared key tree

### A. Key Forest

To address scalability and flexibility in key management, LKH is used as the basis of our scheme. An intuitive solution is to use a key tree for each program as shown in Figure 3(a). However, when user u1 subscribes to two programs simultaneously, he needs to manage two sets of keys in both trees which is not very efficient (see Figure 3(a)). Hence, shared key tree (SKT) is proposed to reduce this cost in key management. As shown in Figure 3(b), we let the two programs share the same sub key tree as represented by the gray triangle. We regroup users so that users subscribing to both programs only need to manage keys in the gray triangle. The advantage of shared key tree is clear: any user subscribing to both g1 and g2 only needs to manage one set of keys for both programs. Moreover, when a user joins or leaves a tree shared by multiple programs, the encryption and communication cost for rekey operations can be significantly less than conventional LKH approaches.

In order to ensure that a user will not pay for subscribed programs multiple times, the key forest obviously should have the following properties, which are guaranteed in any directed and acyclic graph.

Property 3.1: Only one path exists by following the upward links from the root of a tree trs to the DEKs of the programs that share trs;

Property 3.2: Only one path exists by following the upward links from any leaf node in a tree to the root;

Property 3.3: Each user belongs only to one tree in the key forest, and his individual key is the leaf node of the tree.

### B. Root Graph

The root graph in Figure 4 depicts how programs share keys. Since m programs could generate $2^m - 1$ different subscriptions, such a two-layer structure in fact brings two major problems in terms of rekey overheads when the number of programs is large.

First, a program may be included in many subscriptions, which means the DEK of the program is connected with many trees. Assume the DEK is connected with n trees. When a user stops subscribing the program, the DEK needs to be updated and distributed to users in n trees. Because the new DEK is encrypted with the roots of the n trees in rekey, O(n) rekey items are generated. Obviously, if n is large, a leave event results in a huge rekey message. For example, in Figure 5(a),3 programs are included in 5 different subscriptions. Program $g_1$'s DEK $k_{g1}$ is connected with 4 roots $k_{r1}$, $k_{r2}$, $k_{r3}$.

Hence, when $k_{g1}$ is updated due to a leave event, 4 rekey items are needed. To solve this problem, we use a multi-layer structure to connect the DEK with the roots of the shared trees. Second, a subscription is

not a conventional plan that a broadcast service provides, because the subscribed programs of a plan normally cannot be changed by a user. In this paper, users are able to customize the selection of programs in their subscriptions. Thereby, a broadcast service could easily have a large number of different subscriptions. For example, even if a service provides only 30 programs that is a small number in many broadcast services, there could be $2^{30} = 1$ billion different subscriptions, which is much larger than the number of users. Hence, managing keys for all possible subscriptions would overload the server. Now, assume the service has $n$ users and $2^m \gg n$. Although $2^m - 1$ different subscriptions exist, at most $n$ subscriptions are valid, since the number of valid subscriptions cannot be more than the number of users. Hence, this problem can be easily solved by letting the server only manage the valid subscriptions that have at least one user.

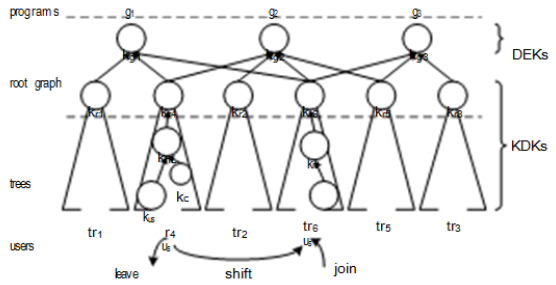


Fig. 4. Key forest

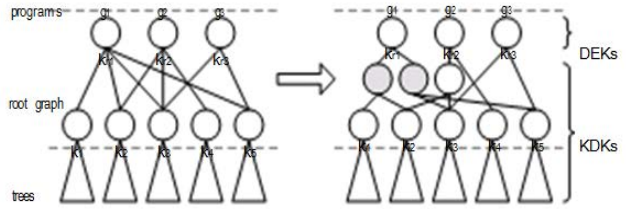


Fig. 5. Multi-layer root graph

## C. Rekey Operations

In this study, we consider user activities of **Joining/leaving/shifting**among trees, instead of joining/quitting/changing among programs. Table I lists the mapping between tree-oriented operations and the corresponding program-oriented user events. Consider the example in Figure 4, where a user $u_s$ shifts from $tr_4$ to $tr_6$. When $u_s$ was in $tr_4$, $u_s$ subscribed $g_1$ and $g_2$. After he shifts to $tr_6$, he subscribes $g_1$, $g_2$ and $g_3$. Hence, the shift in fact means the user adds $g_3$ into his current subscription. Note that the discussion of rekey operations in this study only considers individual user events.

To issue new keys upon a user event, the main task is to identify the keys that need to be changed. We use two types of paths in the key forest to represent the to-be-changed keys. When a user leaves a tree, we say, a **leave path** is formed, which consists of keys that the user will no longer use. When a user joins a tree, we say, an **enroll path** is formed, which consists of keys that the user will use in the future. Similarly, when a user shifts from one tree to another, a leave path and an enroll path are formed. In KTR, a complete path starts from the leaf node and ends at the multiple DEKs of the subscribed programs that share the tree. For example, in Figure 4, when $u_s$ shifts from $tr_4$ to $tr_6$, the leave path consists of $k_{nL}$ and $k_r$, and the enroll path consists of $k_{nj}$ $k_{r6}$, $k_{g1}$, $k_{g2}$ and $k_{g3}$. To broadcast new keys, the server should first compose rekey Packets. In this study, we take the standard LKH approach to Encrypt a new key $k^i$ in a rekey item $\{k^i\}_{kj}$. If $k^i$ is in an enroll path, $k_j$ is the old $k_j$, i.e.$\{k^i\}_k \equiv \{k^i\}_k$ If $k^i$ is in a leave path, $k_j$ is a child key of $k^i$. Readers can refer to [2], [5] for examples of rekey packets.

TABLE I REKEY OPERATIONS

| Tree | Program oriented events. |
|---|---|
| Join a tree | Assume a user has not subscribed to any program. He subscribes to one or multiple programs. |
| Leave a tree | Assume a user has subscribed to several programs. He Unsubscribes to all current programs. |
| Shift among trees | Assume a user has subscribed several programs. He subscribes to one or a few more programs. He unsubscribes to a part of the current programs He changes a part of the current programs. |

KTR to efficiently address the security issue in reusing keys. Since rekey cost is determined by the number of must-be-changed keys, the cost can be minimized if we can find the minimum number of must-be-changed keys when the user joins or shifts to the tree. We name the must-be-changed keys in an enroll path as **critical keys**. KTR changes all keys in a leave path and only the critical keys in an enroll path, while leaving all the other keys unchanged. In this way, the rekey cost can be minimized.

## IV. SHARED KEY MANAGEMENT

In this section, we first present some important concepts in Section IV-A and IV-B, which are used for identifying critical keys . Then, we present the condition under which a key is critical in Section IV-C and IV-D and the corresponding key management algorithms.

### A:Rekey Spots

KTR basically logs how a key was used in rekey messages. We can always find two operations in any rekey message:
1) a key's value is changed or 2) a key is used to encrypt its parent key when the parent key's value is changed. Accordingly, we define two types of spots to log the time points when either operation is committed.

**Algorithm 1** Update of refresh and renew spots Assume $k_i$ is used in the rekey messages upon a user event.

1: **if** $k_i$ is in a leave path **then**
2:renew spots must be added to all $k_i$'s spot series;
3: **end if**
4: **if** $k_i$ is critical in an enroll path **then**
5:renew spots must be added to all $k_i$'s spot series;
6: **end if**
7: **if** $k_i$'s parent key $k_j$ is in a leave path **then**
8:refresh spots must be added to $k_i$'s spot series that are associated with the programs sharing $k_j$ ; 9: **end if**

*Definition 4.1:* Renew spot of a key $k_i$: the time point $t$ when $k_i$'s value is changed. $k_i$'s new value starting from $t$ is denoted as $k_i(t)$.

*Definition 4.2:* Refresh spot of a key $k_i$: the time point $t$ when $k_i$ is used to encrypt its parent key $k_j$ 's new value in a refreshment $\delta(k_j, t; k_i, t)$.

*Definition 4.3:* Refreshment, $\delta(k_j, t; k_i, t')$: a rekey message broadcast at $t$ in the form of $\{k_j(t)\}_{ki(t')}$ $t' \leq t$.

**Algorithm3** Algorithm of KTR in Broadcast Server
1: **if** a join or shift event happens **then**
2:according to TCK, find all critical keys in the tree the user wants to join or shift to;
3:select the best enroll path that has the minimum number of critical keys;
4:change all critical keys in the best enroll path, and broadcast corresponding rekey messages;
5: end if
6: if a leave or shift event happens then
7:change all keys in the leave path, and broadcast corresponding rekey messages;
8: end if
9:update renew, refresh and revive spots according to the latest rekey messages;

Theorem 4.1 indicates that changing only critical keys can ensure past confidentiality. Hence, given a key forest, Algorithm 3 is applied to find the best enroll path and minimize the rekey cost. When a join or shift event happens to a tree, the algorithm uses the depth-first tree traversal approach to find all critical keys in the tree. If a path is found to have fewer critical keys than previously visited paths, the algorithm records it as the best enroll path.

*Corollary 4.1:* When a user joins a tree, a key in the enroll path is a critical key if and only if one of the key's ages is greater than 0.

Before a user joins the tree, his subscription ages for all of the programs sharing this tree are 0. Hence, if the age of a key in the enroll path for this program is greater than 0, the key is older than the user's subscription. According to Theorem 4.1, the key needs to be changed before being distributed to the user.

*Corollary 4.2:* After a user enrolls in a tree, all keys in the enroll path are not older than the user.

According to Theorem 4.1, if a key is older than the user's subscription regarding a program, the key needs to be changed. Hence, at the time when the user enrolls in a tree, the keys, whose ages are older than the user, are renewed and their ages turns to be 0. If the key is not older than the user's subscription regarding any program, the key does not need to be changed. Hence, the key continues to be not older than the user. Therefore, after a user enrolls in a tree, all keys in the enroll path are not older than the user.

*Corollary 4.3:* When a user shifts from a tree to another tree, the keys overlap both trees do not need to be changed.

Assume the user shifts from tree $tr_\alpha$ to tree $tr_\beta$ . According to Corollary 4.2, after the user enrolls in $tr_\alpha$ , all keys in the enroll path cannot not be older than the user. Hence, when the user shifts to $tr_\beta$ , the overlapped keys, which were in the enroll path when user enrolled in $tr_\alpha$ , do not need to be changed according to Theorem 4.1.

### E. Security Analysis

To ensure multicast or broadcast security, group key man- agement should satisfy four security properties [2], [3]: non- group confidentiality, collusion freedom, future confidentiality (forward secrecy), and past

confidentiality (backward secrecy). In the following, we discuss how KTR satisfies these proper- ties.

*Property 4.1:* Non-group confidentiality: passive adver-saries should not have access to any group key.

Because keys are encrypted when being broadcast, passive adversaries can not decrypt any key without knowing decryp -tion key. Hence, KTR obviously satisfies Property 4.1.

*Property 4.2: collision freedom:* by sharing group keys, Multiple present users cannot derive any group key that the they are not holding.

When multiple users collude, they may try to share their keys to derived unknown group keys. the sharing can be rep-presented by a sub graph of the paths belonging to the colluding users. However, in KTR, user does not know any key not in this path. Hence colluding users do not know any key outside the sub graph that represents the collusion. KTR thus satisfies property 4.2 *Property 4.3:* Future confidentiality (forward secrecy): a Leaving user not have access to any group key after leaving his present group. According to Algorithm 3, KTR changes all keys in the leave path, because the leaving user holds these keys. Hence, the leaving user will not have the new keys after the user leaves his group. KTR thus satisfies Property 4.3.

*Property 4.4:* Past confidentiality (backward secrecy): a Joining user added at time t should not have access to any keys used to encrypt data before t.According to Algorithm 3, KTR changes all critical keys in the enroll path when a user joins. Theorem 4.1 basically proves that the joining user can only derive past group keys from critical keys. Hence, changing critical keys and reusing non-critical keys prevent the joining user from obtaining past group keys. KTR thus satisfies Property 4.4.

## V. CONCLUSION

In this work, we investigated the issues of key management in support of *secure* wireless broadcast services. We proposed KTR as a scalable, efficient and secure key management approach in the broadcast system. We used the key forest to exploit the overlapping nature between users and programs in broadcast services. KTR let multiple programs share a single tree so that the users subscribing these programs can hold fewer keys. In addition,we proposed an ovel shared key management approach to further reduce rekey cost by identifying the minimum set of keys that must be changed to ensure broadcast security. This approach is also applicable to other LKH-based approaches to reduce the rekey cost as in KTR. Our simulation showed that KTR can save about 45% of communication overhead in the broadcast channel and about 50% of decryption cost for each user, compared with the traditional LKH approach.

## REFERENCES

[1] J. Xu, D. Lee, Q. Hu, and W.-C. Lee, “Data broadcast,” in Handbook of wireless Networks and Mobile Computing, I. Stojmenovic, Ed. NewYork: John Wiley and Sons, 2002, pp. 243-265.

[2] D. Wallner, E. Harder, and R. Agee, “Key management for multicast:issues and architectures,” IETF RFC 2627, 1999.

[3] J. Snoeyink, S. Suri, and G. Varghese, “A lower bound for multicast keydistribution,”inIEEEInfocom,vol.1,2001,pp.422-431

[4] S. Mittra, “Iolus: a framework for scalable secure multicasting,” in ACMSIGCOMM, vol. 277-288, 1997.

[5] C.K.Wong,M.Gouda,and S.S.Lam,“Secure group communicationsusing keygraphs,”inACMSIGCOMM,1998,pp.68-79.

[6] Y. Kim, A. Perrig, and G. Tsudik, “Simple and fault-tolerant key agreement for dynamic collaborative groups,” in ACM CCS, 2000, pp. 235-244.

[7] S. Setia, S. Koussih, S. Jajodia, and E. Harder, “Kronos: a scalablegroup re-keying approach for secure multicast,” in IEEE Symposium onSecurity and Privacy, 2000, pp. 215-228.