KTR: an Efficient Key Management Scheme For Air Access Control

Qijun Gu Peng Liu Wang-Chien Lee Chao-Hsien Chu

Pennsylvania State University, University Park, PA 16802
qgu@ist.psu.edu, pliu@ist.psu.edu, wlee@cse.psu.edu, chu@ist.psu.edu

Abstract

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. 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 conventional approaches based on logical key hierarchy.

1 Introduction

In wireless broadcast services, data items are grouped into *programs*. The set of programs a user subscribes is called the user's *subscription*. However, any user can monitor and log the broadcast data, if the data is not encrypted. Symmetric-key-based encryption is a natural choice to ensure secure access of data on air. Each program has one key to encrypt the data items. The key is issued to the user who is authorized to decrypt the data items. If a user subscribes multiple programs, it needs keys for all of these programs. Nevertheless, a critical issue remains, i.e. *how can we efficiently and flexibly manage the keys when a user joins/leaves/changes the service without compromising security and interrupting operations of other users?* In broadcast services, a user may subscribe programs of interests, quit the service, or change his subscription at any time. Since a user may subscribe multiple programs, it is in fact inefficient for the user to hold separate sets of keys for each subscribed program. In addition, when a user changes his subscription (i.e. add or unsubscribe some programs), we argue that it is unnecessary to change keys for the unchanged programs as long as security can be ensured. Hence, the primary design goal of our proposal for key management in wireless broadcast services is to provide flexibility, efficiency and security simultaneously.

Although there are a great number of existing studies on key management in the literature of group communication, they do not meet our requirements in broadcast services (refer to Section 2 for more discussion). Thereby, we propose a new key management scheme, namely *key tree reuse* (**KTR**), based on two important observations: (1) users who subscribe multiple programs can be captured by a shared key tree, and (2) old keys can be reused to save rekey cost without compromising security when users changes their subscriptions. To support subscription of multiple programs simultaneously, KTR exploits the overlapping among different subscriptions to let multiple programs share the same set of keys. Hence, our KTR scheme inherently enables users to handle less keys and thus reduces the demands of storage and processing power in resource-limited mobile sets. We also find that, in many circumstances, it is unnecessary to change the shared keys of the programs that a user joins or adds. KTR thus provides a novel approach to efficiently handle rekeying of the shared keys by identifying the minimum number of keys that must be changed to ensure security and minimize rekey costs regarding possible subscriptions. Our finding can also be employed in traditional key management schemes (e.g. *logical key hierarchy* (**LKH**) [2, 3]) to improve their performance.

2 Related Works

The most widely applied approach in secure multicast is logical key hierarchy (LKH) [2, 3], which is a key tree applied in a multicast group. Nevertheless, directly applying LKH in broadcast services is not suitable. A broadcast service provides many programs, each of which is equivalent to a multicast group. A user subscribing multiple pro-

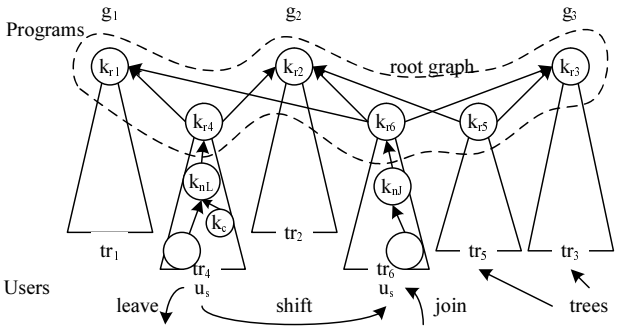


Figure 1. Key Forest

grams need to hold multiple sets of keys. Hence, the idea to let multiple programs share the same key tree is more efficient [1]. However, we find that only reducing the redundancy of key trees is still not enough to address security and efficiency in broadcast services. In fact, new problems are raised when a user changes his subscription. We need to determine whether keys for the programs the user keeps should be changed to ensure security instead of simply renewing keys as in traditional LKH-based approaches. Consequently, KTR differs from the key sharing approach [1] in that it has a unique approach to handle the shared keys at the minimum rekey cost without compromising security. Our extensive simulations show that KTR can better improve the performance.

3 Key Tree Reuse (KTR)

Programs share keys as depicted in Figure 1. In the graph, each tree, structured as LKH, represents a set of users having the same subscription. A key tree is shared by a set of programs belonging to the subscription. The root of a shared key tree (**SKT**) is a KDK. All roots form a *root graph*, and all trees form a forest. If a tree is shared by several programs, its root is connected to the roots of these programs, which are DEKs of these programs. Obviously, if a user is in a SKT, he only handles one set of keys for all programs that connect the SKT.

When keys are shared by multiple programs, problems are raised if a user joins or shifts to a tree in which some programs have already been subscribed by the user. Still consider the example in Figure 1 where a user shifts from tr_4 to tr_6. It is noticed that after shifting to tr_6, the user is still subscribing programs g_1 and g_2 as he was in tr_4. Obviously, if a key (for instance k_{r_1} or k_{r_2}) is only used by g_1 and g_2, there is no need to change the key. Nevertheless, do we need to change keys (for instance k_{r_6} and k_{n_J}) in tr_6, which are shared by g_1, g_2 and g_3?

The idea to handle a shared key is to inspect whether the key was previously used in encrypting its parent key. In LKH, a key k_i may be used to encrypt its parent key k_j, when k_j is in a leave path and needs to be updated. The encryption takes the form $\{k_j\}_{k_i}$ [3]. To ensure security, a lazy approach is to change the shared keys as in LKH, no matter how they were used. However, this approach brings more rekey costs to mobile devices. Hence, we propose a special approach in KTR to efficiently address the security issue. The challenge in such a shared key management is how to decide whether a key has ever been directly or indirectly used in a way that may compromise the security. 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 the tree. We name the must-be-changed keys in an enroll path as **critical keys**. In another word, in an enroll path, KTR only changes critical keys, while leaving other keys unchanged, and thus the rekey cost can be reduced to the minimum. Note that keys in a leave path always need to be changed to ensure forward secrecy [2].

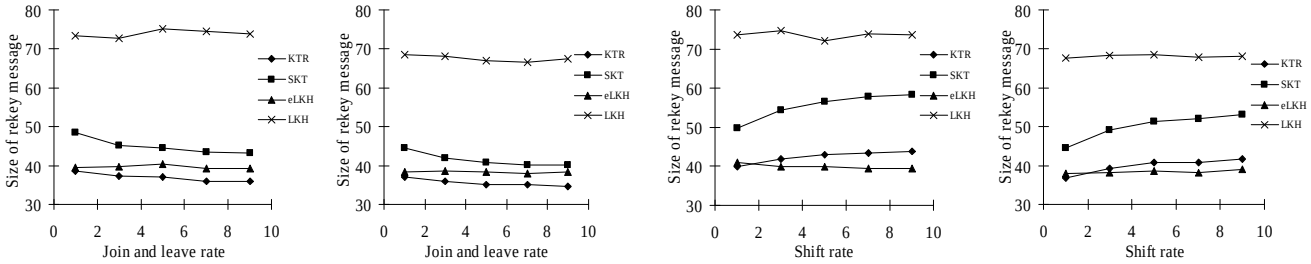
Table 1. Cases in key management

Case	Major subscriptions	Major events
Case I	Multiple	Join and leave
Case II	Single	Join and leave
Case III	Multiple	Shift
Case IV	Single	Shift

4 Performance Evaluation

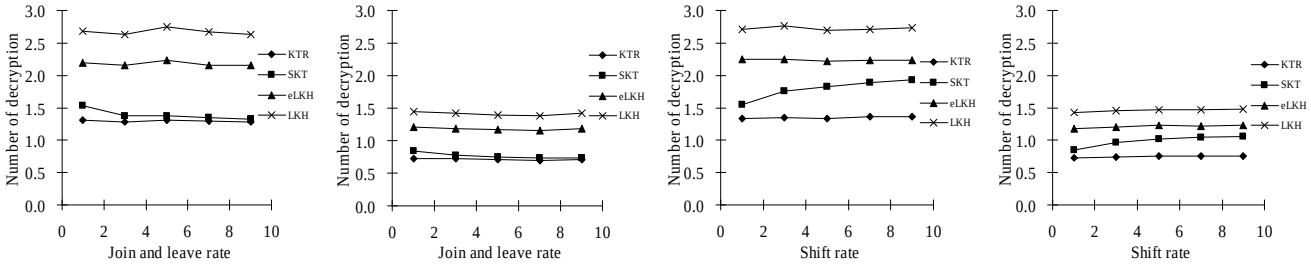
Shared key tree and *critical key* are the two important ideas in KTR. We conduct a simulation-based performance evaluation to examine their impacts. Four representative schemes (KTR, SKT, eLKH, LKH) are compared. SKT is the approach in [1] where only shared key tree is applied. eLKH is an approach where only critical key is applied to enhance LKH. We assume that the server provides 5 programs and 10000 subscribers (on average). The key forest consists of 31 trees when shared key tree is adopted. Each tree represents a unique option of subscriptions. We vary the shift rate and the join/leave rate separately in order to observe their impacts on the rekey performance. The result of our performance comparison is obtained by averaging the rekey cost over 3000 random user events. Four test cases are generated for the evaluation based on major subscriptions and major events (summarized in Table 1).

This evaluation is to examine the overhead of KTR on resource limited mobile devices in terms of communication, storage, power consumption, computation, etc, which can be captured by two important metrics, *average rekey message size per event* and *average number of decryption per event per user*. The former, measured as the number of en-



(a) Case I (b) Case II (c) Case III (d) Case IV

Figure 2. Average rekey message size per event



(a) Case I (b) Case II (c) Case III (d) Case IV

Figure 3. Average number of decryption per event per user

cryptions $\{*\}_k$ in the rekey message. The latter measures computation cost and power consumption in a mobile device.

The comparison of SKT and eLKH brings some very good insights regarding to the impacts of our two proposed ideas on the performance metrics. Our experiments shows that critical key is a more important factor in reducing the rekey message size than shared key tree (see Figure 2). On the other side, shard key tree is a dominant factor in reducing the number of decryption (see Figure 3). In summary, KTR combines the advantages of both ideas of shared key tree and critical key, and thus is the most efficient scheme for key management in secure wireless broadcast systems. It has a light communication overhead (i.e. its average rekey message size per event is the least or close to the least among all schemes). Meanwhile, it incurs less computation and power consumption on mobile devices than the other schemes (i.e. its average number of decryption per event per user is the smallest among all solutions).

5 Conclusion

In this paper, we investigate the issues of key management in support of *secure* data access in wireless broadcast services. To the best knowledge of the authors, this is the first research to address key management in wireless broadcast services, since simply applying other techniques is not suitable. We propose KTR as a flexible, scalable, efficient and secure key management approach in the broadcast system. We use the shared key structure to allow users who subscribe multiple programs to hold less keys. In addition, we propose an approach in KTR for shared key management. The approach reduces rekey cost by identifying the minimum set of critical 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 shows 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.

6. Acknowledgement

This work was partially supported by NSF ANI-0335241, NSF CCR-0233324, and Department of Energy Early Career PI Award.

References

[1] Y. Sun and K. R. Liu. Scalable hierarchical access control in secure group communications. In *IEEE Infocom*, 2004.

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

[3] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. In *ACM SIGCOMM*, pages 68–79, 1998.