

Adding Reliable and Self-Healing Key Distribution to the Subset Difference Group Rekeying Method for Secure Multicast

Sencun Zhu, Sanjeev Setia, and Sushil Jajodia

Center for Secure Information Systems, George Mason University, Fairfax, VA 22030.
{szhu1, setia, jajodia}@gmu.edu

Abstract. We study two important issues related to the Subset Difference Rekeying (SDR) method [4]. First, we present a reliable key distribution scheme, called WFEC-BKR, that enables members to receive the current group key in a reliable and timely fashion inspite of packet losses in the network. Through simulation, we show that in most scenarios, WFEC-BKR outperforms previously proposed schemes for reliable rekey transport. Second, we present a group key recovery scheme that adds the self-healing property to SDR, i.e., our scheme enables a member that has missed up to a certain number m of previous rekey operations to recover the missing group keys without asking the key server for retransmission. The additional communication overhead imposed by our key recovery scheme is quite small (less than $3m$ additional keys).

1 Introduction

In recent years, many approaches for scalable group rekeying have been proposed, e.g. LKH [8, 9, 11], OFT [1], MARKS [2], Subset Difference [4] and self-healing [6]. Further, it has been proposed that groups be re-keyed periodically instead of on every membership change [5, 10]. Periodic or batched rekeying can reduce both the processing and communication overhead at the key server, and improve the scalability and performance of key management protocols based on logical key trees.

In addition to the rekeying algorithm, the communication overhead of group rekeying also depends on the protocol used for reliably delivering the updated keys to the members of the group. Recently, researchers have proposed customized reliable multicast protocols for group rekeying, e.g., Proactive-FEC [10] and WKA-BKR [7], which take advantage of the special properties of the rekey payload for achieving reduced communication overhead in comparison to conventional reliable multicast protocols.

Among the rekeying protocols proposed in the literature, the Subset Difference Rekeying method (SDR) [4] is one of the few protocols that have the property of *statelessness*. In a stateless rekeying protocol, in order to decode the current group key, a member only needs to receive the keys that are transmitted by the key server during the current rekey operation. This property makes SDR very attractive for secure multicast applications where members may go off-line frequently or experience burst packet losses. Furthermore, SDR has been shown to be very efficient in terms of communication overhead.

In this paper, we study two important issues related to the key delivery protocol used for the SDR method. First, we address the issue of reliable rekey transport for SDR. We present a key distribution scheme, called WFEC-BKR, that enables members to receive the current group key in a reliable and timely fashion despite the presence of packet losses in the network. WFEC-BKR is a hybrid protocol that combines the advantages of two previously proposed rekey transport protocols – the proactive FEC based key delivery protocol [10] and the WKA-BKR protocol [7]. Through simulation, we show that in most scenarios, WFEC-BKR outperforms the other rekey transport protocols.

Second, we examine the issue of *self-healing* group key distribution for SDR. We present a key recovery scheme that adds the self-healing property to SDR, i.e., the scheme enables a member that has missed up to a certain number m of previous rekey operations to recover the missing group keys without asking the key server for retransmission. This self-healing key recovery property results in reduced network traffic and also reduces the load on the key server, and is especially useful for group members that may experience burst packet losses. Through a detailed simulation, we found that the communication overhead imposed on the key server by our recovery scheme is quite small (less than $3m$ additional keys).

The remainder of this paper is organized as follows. In Section 2, we discuss related work and introduce the SDR method in more detail. In Section 3 we present our hybrid reliable key distribution scheme and evaluate its performance

through detailed simulation. Section 4 describes our key recovery schemes and its performance. Finally, we summarize our work in Section 5.

2 Related Work

The group rekeying protocols proposed in the literature can be divided into stateful and stateless protocols. The stateful class of protocols includes several protocols based upon the use of logical key trees, e.g., LKH [8,9] and OFT [1]. In these protocols, a member must have received all the key encryption keys of interest in all the previous rekey operations; otherwise, it will be unable to decode the new (group) key being transmitted during the current rekeying operation, and will have to ask the key server to retransmit any keys it is missing. Among these protocols, neither LKH nor OFT includes any mechanisms for reliable key distribution. To address the issue of reliable key delivery for these group rekeying approaches, researchers have proposed protocols based on the use of proactive redundancy such as the proactive-FEC based key delivery protocol [10] and WKA-BKR [7]. However, these protocols only address the issue of reliable key delivery for the *current* rekeying operation.

Stateless group rekeying protocols form the second class of rekey protocols. In these protocols, a legitimate user only needs to receive the keys of interest in the current rekey operation to decode the current group key. In other words, there is no dependency between the keys used in different rekeying operations. One such protocol is the subset difference rekeying method (SDR) presented by Naor *et al* [4]. In this paper, we focus on key delivery protocols for SDR. Hence, in Section 2.1, we describe SDR in more detail.

Another example of a stateless protocol is the self-healing key delivery protocol proposed by Staddon *et al* [6]. In addition to statelessness, this protocol has the property (referred to as self-healing) that a group member that has not received a previous group key (due to network packet loss) can recover that group key *on its own* without contacting the key server. The self-healing protocol is based on polynomial-based secret sharing techniques. We note that this protocol has two limitations that may discourage its deployment for some applications. First, in this protocol, an application's lifetime is pre-divided into a certain number of sessions, and the key server initiates a group rekeying at the beginning of each session. Thus, this protocol is not suitable for applications which have a security requirement of immediate user revocation. Second, the maximum allowed number of revoked users during these sessions has to be pre-determined and must not be exceeded for the protocol to be secure. In Section 4, using techniques such as one-way key chains, we propose a protocol that adds the self-healing property to the SDR method with very small additional overhead. In addition, our protocol can perform immediate group rekeyings and there is no upper bound on the number of revoked users.

We note that periodic batched rekeying has been shown to improve the scalability and performance of group key management protocols [5, 10]. As such, in this paper we discuss our key delivery schemes for SDR in the context of periodic batched rekeying. (Note, however, that our key delivery schemes can be used for both individual and batched rekeying.)

2.1 Subset Difference Rekeying Method

In SDR, during a rekey operation the key server partitions the current members of the group into a minimal number of subsets, and then encrypts the new group key with the common key of each subset separately. Hence, the number of encrypted keys to be distributed to the users is the same as the number of subsets the method generates.

Figure. 1 illustrates a subset S_{ij} in SDR. The users are viewed as leaves in a complete binary tree. The subset S_{ij} can be thought as the set of users in the subtree rooted at node V_i minus the set of users in the subtree rooted at node V_j . More generally, a valid subset S is represented by two nodes in the tree (V_i, V_j) such that V_i is an ancestor of V_j . A leaf u is in $S_{i,j}$ iff it is in the subtree rooted at V_i but not in the subtree rooted at V_j . The subset S_{ij} is associated with a unique key only known by the users in S_{ij} . We refer the reader to [4] for a description of the key assignment algorithm.

Let N be the set of all users and R the set of revoked users. Let Cover denote the subset cover, i.e., the collection of disjoint subsets $S_{i1,j1}, S_{i2,j2}, \dots, S_{im,jm}$ which partitions $N \setminus R$. Let T be the backbone tree induced by the set R of vertices and the root, i.e., the minimal subtree of the full binary tree that connects all the leaves in R . The algorithm for finding the subset cover is as follows. Consider maximal chains of nodes with outdegree 1 in T . More precisely,

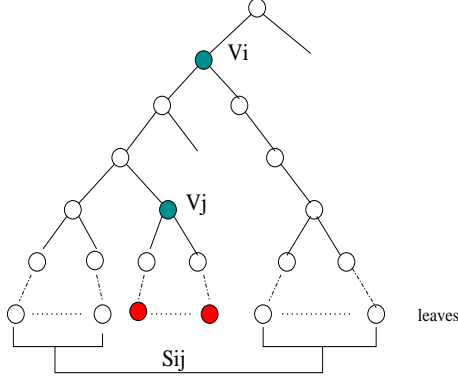


Fig. 1. The Subset Difference Rekeying Method. The solid leaf nodes denote the revoked users. Subset S_{ij} contains the current members.

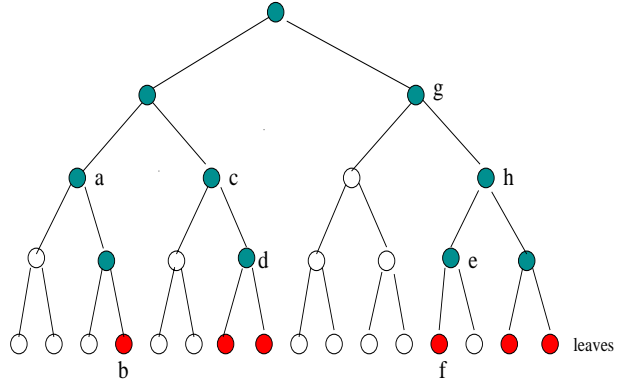


Fig. 2. An example of SDR where all the solid nodes form the backbone tree T .

each such chain is of the form $[v_{i1}, v_{i2}, \dots, v_{il}]$ where (i) all of $v_{i1}, v_{i2}, \dots, v_{il-1}$ have outdegree 1 in T (ii) v_{il} is either a leaf or a node with outdegree 2 and (iii) the parent of v_{i1} is either a node of outdegree 2 or the root. For each such chain where $l \geq 2$ add a subset $S_{i1,il}$ to Cover. Note that all nodes of outdegree 1 in T are members of precisely one such chain.

In Fig. 2, we show an example of the key tree in SDR. The solid leaves corresponds to all the revoked nodes in R , and all the solid nodes form the backbone tree T that connects all these leaves. Based on the above algorithm, we can easily get all the subsets, $\{S_{ab}, S_{cd}, S_{ef}, S_{gh}\}$, because the nodes a, c, e, g have outdegree of 1 while their parents have outdegree of 2, and b, d, f, h either are leaf nodes or have outdegree of 2.

Naor *et al* [4] show that the average number of subsets in the subset cover is $1.25r$ when there are r revoked users in R . Thus, the communication complexity (i.e., the number of subsets) is independent of the group size, which makes this algorithm very scalable, particularly when $r \ll |N|$ (Chen and Dondeti [3] show that SDR performs better than LKH for rekeying of large batches). The number of keys stored by each user is $0.5 \log^2 |N|$.

In this algorithm, a current group member u only needs to receive exactly *one* encrypted key in every rekeying, which is the new group key encrypted with the key of a subset to which it belongs. Because user u is provided with the keys for all the subsets it might belong to at the time it joins the group, the key encryption keys used in each rekey operation are independent of each other, leading to the stateless nature of the protocol. Note that in SDR the key tree does not expand or shrink when users join or depart so that the keys a user obtains from the key server at its joining time never need to be updated. Therefore, the key tree maintained by the key server has to be large enough to hold all the potential users that may join the group during the lifetime of the application. Further note that in every rekeying, the key server considers all the leaf nodes that have not been mapped to new users as revoked nodes when it computes the subset cover, for the purposes of backward confidentiality.

3 Reliable Key Delivery for SDR

In a group rekeying, the key server first executes the SDR algorithm to determine the subset cover for all the current members of the group. Then it generates a new group key, and encrypts the group key separately with the common key of each subset in the subset cover. Finally, it multicasts all the encrypted keys to the group. The question we wish to address is: how can every on-line user receive the new group key in a reliable and timely fashion in the presence of packet losses in network?

Previous work on the problem of reliable key delivery such proactive FEC based key delivery protocol [10] and WKA-BKR [7] has focussed on the LKH algorithm and not on SDR. In this section, we first discuss the application of these approaches to SDR and then present an improved hybrid approach called WFEC-BKR.

Proactive FEC-based Key Delivery In the proactive FEC-based approach [10], the key server packs the encrypted keys into packets of s_k keys. These packets are divided into FEC blocks of k packets. The key server then generates

$\lceil(\rho - 1)k\rceil$ parity packets for each block based on Reed Solomon Erasure (RSE) correcting codes, where $\rho \geq 1$ is the pro-activity factor. A user interested in the packets from a certain block can recover all the original packets in the block as long as it receives any k out of $\lceil k\rho\rceil$ packets from the block. If a user does not receive a packet that contains the encrypted key of interest to it, but it receives t ($t < k$) packets from the block that contains this packet, it will ask the key server for retransmission of $k - t$ new parity packets. The key server collects all the retransmission requests, and then for each block it generates and transmits the *maximum* number of new parity packets required by users. The retransmission phase continues until all the users have successfully received their keys.

WKA-BKR The WKA-BKR scheme [7] uses a simple packet replication technique in which packets are transmitted multiple times, but it takes advantage of two properties of the rekey transport payload to minimize the overall bandwidth overhead of a rekey operation. First, the encrypted keys are assigned different replication weights, depending on the number of users interested in them and the loss rates of these users. Clearly, when a subset in the SDR method covers a larger number of users or these users have higher loss rates, the encrypted key for this subset should be given a higher degree of replication so that most of these users will receive the key reliably. Hence, in this scheme the key server first determines the weight w_i for each encrypted key K_i based upon the users interested in that key. It then packs the keys that have the same weight $\lfloor w_i \rfloor$ into the set of packets p_i . When broadcasting the packets, the key server sends packets in p_i $\lfloor w_i \rfloor$ times. This process is called weighted key assignment (WKA). Second, during the retransmission phase, since each user that has made a retransmission request only needs one encrypted key to decode the current group key, there is no need for the key server to retransmit the entire packet sent in the previous round that contained the requested key. Instead, the key server repackages the keys that need to be retransmitted into new packets before retransmitting them. This process is called batched key retransmission (BKR). The WKA-BKR scheme has been shown to have a lower bandwidth overhead than the other schemes in most scenarios.

3.1 WFEC-BKR: A Hybrid Approach

In our comparative performance evaluation of the proactive-FEC based scheme and the WKA-BKR scheme [7], we found that one reason that WKA-BKR has a lower bandwidth overhead than the proactive FEC-based approach is due to the bandwidth efficiency of its retransmission scheme, i.e., BKR. In the proactive FEC based approach, the key server retransmits the *maximum* number of required parity packets for each block. Therefore, the bandwidth overhead is dominated by retransmissions due to users experiencing high packet losses. On the other hand, we found that the proactive FEC-based approach usually has a smaller bandwidth overhead than WKA-BKR in the first round of transmission, especially when the weights of many packets in WKA are larger than 2. This is because RSE encoding used in FEC is more efficient than the simple replication used in WKA. Further, we found that proactive FEC-based protocols have a lower latency of key delivery than WKA-BKR. Based on these observations, we propose a hybrid scheme, called WFEC-BKR, which is a combination of weighted FEC and BKR.

In WFEC-BKR, the key server first packs the encrypted keys into s_p packets of s_k keys, and then divides the packets into s_b blocks of k packets. The key server then chooses an appropriate ρ and generates $\lceil(\rho - 1)k\rceil$ parity packets for each block, based on the analysis in Section 3.2. Finally, it broadcasts all the packets. In the retransmission phase, a user that has not received its key reports the missing key. The key server collects all the retransmission requests, repackages the requested keys into new packets, and then broadcasts these packets. This process is repeated until all users have received their keys successfully.

3.2 Determining The Proactivity Factor ρ

We note that, unlike the proactive FEC-based protocol discussed in [10], in our protocol the proactivity factor for different FEC blocks will typically be different depending upon the number of users interested in the keys in a particular block as well as the estimated packet loss rates of these users.

Now we consider the issue of selecting an appropriate proactivity factor ρ for each FEC block. Clearly, increasing ρ results in reduced latency at the expense of increased bandwidth overhead. We now sketch an approach that can be used to determine the ρ for each block, based on the number of users interested in the keys in each block and the loss probabilities of these users (a user can estimate its packet loss rate and piggyback this information in the NACK it

sends to the key server.). Thus, we can use this approach to obtain a reasonable initial set of parameters (ρ and k) that can balance the bandwidth overhead with the key delivery latency.

Consider a user u_i with a loss probability p_i . Recall that each user is interested in exactly one key (say K) out of the keys transmitted by the key server. Let ρ_x be the proactivity factor for the FEC block B_x that contains K and let k denote the FEC block size. Let $L_x = \lceil k\rho_x \rceil$. The probability that u_i receives the packet that contains K is $1 - p_i$. In the event it does not receive the packet that contains K , it is still able to reconstruct that packet if it receives at least any k other packets out of the L_x packets in the FEC block. Therefore, the probability $p(x, i)$ that it will receive K in the first round of transmission is

$$p(x, i) = (1 - p_i) + p_i \left(\sum_{j=k}^{L_x-1} \binom{L_x-1}{j} (1 - p_i)^j (p_i^{L_x-j-1}) \right). \quad (1)$$

Using this approach, the key server can compute the probabilities $p(x, i)$ for all the users u_i interested in the key K . By repeating this process for all the keys in a block, adding together all the probabilities $p(x, j)$, where u_j is a user interested in one of the keys in the block B_x , and dividing the sum of the probabilities by the total number of users interested in the keys in the block, we can obtain the expected value of f_x , the fraction of users interested in the keys in block B_x that will receive their keys in the first round of the key delivery protocol.

3.3 Performance Evaluation

Metrics and Simulation Model In this section, we evaluate the performance of WFEC-BKR scheme by comparing it to that of the proactive FEC and the WKA-BKR schemes. We use two metrics in this evaluation: (i) the *average bandwidth overhead* at the key server, defined as the ratio of the total bandwidth (including the bandwidth of the original rekey payload and the replicated and the retransmitted packets) to the bandwidth of the original rekey payload, and (ii) the *fraction of members who successfully receive the group key in the first (transmission) round* of the key delivery protocol. The second metric reflects the latency of group rekeying.

The results are obtained via simulation using a heterogeneous network packet loss model in which a fraction $\alpha = 20\%$ of the receivers have a high packet loss probability $p_h = 0.2$, whereas the remaining receivers have a low packet loss probability $p_l = 0.02$. The packet loss a user experiences is assumed to be independent. We examine the performance of the schemes using the following group characteristics. The key server constructs a binary key tree of height 13, knowing that the maximum number of users who join the group over the lifetime of the application will not exceed $2^{13} = 8092$. Initially, we assume that there are 5000 users in the group. At each rekeying event, the group membership of 100 (randomly selected) users is revoked and 100 new users join the group.

We use the method of independent replications for our simulations and all our results have 95% confidence intervals that are within 1% of the reported value. In our discussion below, we use $FEC(x)$ to denote the proactive FEC based key delivery protocol [10] with pro-activity factor x , and $WFEC(x)$ -BKR to denote the hybrid scheme which uses pro-activity factor x for the first round transmission and then uses BKR for retransmission.

Results

Bandwidth Overhead Fig. 3 plots the rekeying bandwidth overhead (y-axis) of the schemes for 30 consecutive rekeying events (x-axis). We can make the following observations from this figure. First, overall, the hybrid scheme WFEC(1.2)-BKR has the smallest bandwidth overhead, whereas FEC(1.2) has the second largest bandwidth overhead (only slightly lower than FEC(1.6)). The difference between WFEC(1.2)-BKR and FEC(1.2) is significant. Since WFEC(1.2)-BKR and FEC(1.2) have the same bandwidth overhead for replication in the first round of transmission, the difference indicates that batched key retransmission is more efficient than retransmitting the maximum number of required parity packets for each block. Note that WFEC(1.6)-BKR outperforms FEC(1.6) due to the same reason.

Second, the WKA-BKR scheme has a high bandwidth overhead at the beginning of the simulation, but the overhead decreases with each rekeying event. In stateful protocols such as LKH, the bandwidth overhead of WKA-BKR and FEC is mainly a function of the group size and the number of joins and leaves being processed as a batch [10, 7]. Since each rekeying event in our simulation has the same number of member joins and leaves, we would not expect there to be much variation in the bandwidth overhead for different rekeying events for LKH. In contrast, for SDR the

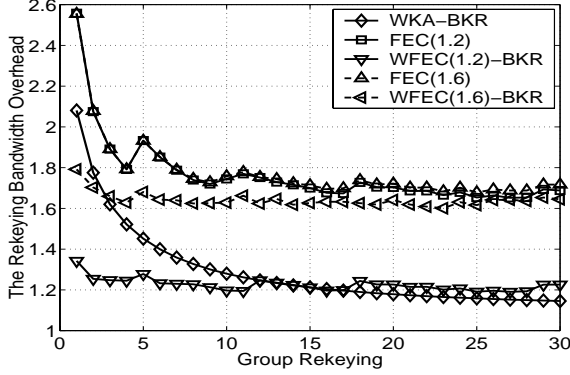


Fig. 3. Key server bandwidth overhead for different rekeying events.

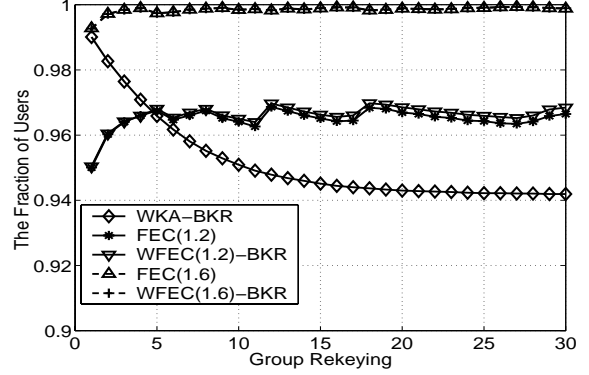


Fig. 4. The fraction of members who receive their keys in the first round.

bandwidth overhead of WKA-BKR depends upon the sizes of the subsets in the subset cover. The composition of the subset cover tends to change over time. For example, the subset to which a user belongs is split when another user from the same subset is revoked and this results in two smaller subsets. As more users leave the group, most subsets become very small; as a result, their replication weights under WKA-BKR are also reduced, leading to a lower overall bandwidth overhead.

Third, the curves for FEC fluctuate at some rekeying points while the curve for WKA-BKR is very smooth. This fluctuation arises from the use of different FEC block sizes k for different rekeying events in order to minimize the overhead of packet padding.

Latency In Fig. 4 we plot the fraction (f_r) of users who receive the group key in the first (transmission) round of the key delivery protocol for 30 consecutive rekey events. We observe that FEC(1.2) and WFEC(1.2)-BKR have the same latency as do FEC(1.6) and WFEC(1.6)-BKR. However, f_r is larger than 0.99 for FEC(1.6) and WFEC(1.6)-BKR, whereas it is around 0.97 for FEC(1.2) and WFEC(1.2)-BKR. This is not surprising since the degree of redundancy in FEC(1.6) is larger than that in FEC(1.2). For WKA-BKR, f_r decreases over time since the replication weights used by the WKA algorithm tend to be reduced as discussed above.

Overall, from Figures 3 and 4, we can conclude that WFEC-BKR has low bandwidth overhead (comparable to that of WKA-BKR), and relatively low latency (comparable to that of proactive FEC-based key delivery).

4 Self-Healing Key Delivery for SDR

The reliable key delivery protocols discussed in Section 3 work well for scenarios where a user experiences random packet losses. However, a user might have to request multiple packet retransmissions until it finally receives the encrypted key of interest to it. There is no guarantee that it will receive the group key before the next group rekeying event. This is especially true for users that are experiencing intermittent burst packet losses. Another similar scenario arises when a user is off-line (while still a member of the group) at the time of group rekeying. If the user receives data that was encrypted using a group key that it has not received, it will need to obtain that group key.

A self-healing key delivery protocol allows a user to obtain missing group keys on its own without requesting a retransmission from the key server. This is accomplished by combining information from the current key update broadcast with information received in previous key update broadcasts. In this section, we will discuss two schemes that add the self-healing property to SDR. We say a scheme has m -recoverability if the maximum number of previous group keys a legitimate user can recover is m .

4.1 Scheme I: The Basic Scheme

Figure 5 shows a sequence of rekeying events. Let $T(i)$ be the current rekeying time, and $K(i)$ is the new group key to be distributed. A simple approach that enables a current member to recover the previous m group keys, i.e.,

$K(i-m), K(i-m+1), \dots, K(i-2), K(i-1)$, is to encrypt these m keys with the current group key $K(i)$ individually and broadcast them to the group. Hence, as long as a user receives $K(i)$ reliably (e.g., through WFEC-BKR), it will be able to recover the previous m keys. However, this approach does not enforce *backward confidentiality* [9], because a newly joined user can also recover these keys.

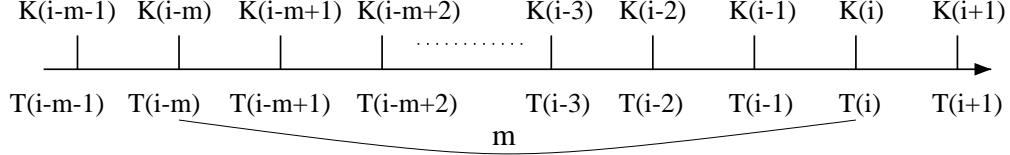


Fig. 5. Recovering the previous group keys, here $T(i)$ is the current rekey time.

To solve this problem, it is important to bind the time at which user joined the group with its ability to recover a previous group key. In other words, a user should only allowed to recover the group keys that were used after it joined the group. To achieve this goal, in our scheme the key server encrypts each group key, $K(i-j)$, $1 \leq j \leq m$ with a key that is derived by XORing the current group key $K(i)$ with the group key $K(j-1)$. Thus, the key server broadcasts m encrypted keys as follows:

$$\text{KeyServer} \longrightarrow * : \{K(i-m)\}_{K(i-m-1) \oplus K(i)}, \{K(i-m+1)\}_{K(i-m) \oplus K(i)}, \dots, \\ \{K(i-1)\}_{K(i-2) \oplus K(i)},$$

where \oplus is the XOR operation and $\{K2\}_{K1}$ means encrypting key $K2$ with key $K1$. A user that joined the group at time $T(j)$, $i-m < j < i$, and received $K(j)$ can recover all the keys between $K(j)$ and $K(i)$ after it receives $K(i)$. A newly joined user, i.e., a user joining at $T(i)$, cannot recover the previous keys because it does not hold any previous keys. On the other hand, a user that was off-line for more than m rekeying periods cannot recover these previous keys. The communication cost is m keys for this basic scheme.

However, there are two scenarios where the security of this scheme may be compromised. The first scenario arises when a revoked user joins the group again some time in the future. For instance, a user that left the group at $T(j)$, $i-m < j < i$, and rejoins at $T(i)$ will be able to recover all the keys between $K(j)$ and $K(i)$ based on $K(j-1)$ and $K(i)$. A similar scenario arises when a user that has left the group or whose membership was revoked colludes with a newly joined user. In the above example, a user whose membership was revoked at time $T(j)$ and has the key $K(j-1)$ could collude with a newly joined user who has $K(i)$ to recover the intermediate keys that they are not authorized to have.

4.2 Scheme II: Dealing with the Rejoining/Colluding Attack

We now propose an extension to the basic scheme that addresses the rejoining/colluding attack described above. The key idea is to bind the ability of a user to recover a previous group key not only to the time at which it became a member but also to its membership duration. The scheme involves the following steps.

1. In each group rekeying, the key server generates a key chain of size $m+1$. Let the keys in the key chain generated for the rekeying at $T(i)$ be $K^m(i), K^{m-1}(i), \dots, K^1(i), K^0(i)$, where $K^0(i) = H(K^1(i)) = H^2(K^2(i)) = \dots = H^m(K^m(i))$ and H is a one-way hash function such as SHA-1. Due to the one-wayness of the hash function, a user knowing $K^j(i)$ can compute all the keys $K^{j-1}(i), \dots, K^0(i)$ independently, but it cannot compute any of the keys $K^{j+1}(i), \dots, K^m(i)$. $K^0(i)$ is the group key that all the users should use for data encryption between $T(i)$ and $T(i+1)$.
2. The users in the group are considered to be partitioned into $m+1$ subgroups, depending upon their membership duration. Each subgroup is associated with a separate key from the one-way key chain generated in the first step. Specifically, $K^j(i)$ is the key intended for the members that joined the group at $T(i-j)$ for $0 \leq j < m$, and $K^m(i)$ is the key intended for members that joined at or before $T(i-m)$. The algorithm used for key distribution is discussed in more detail later in this section.

3. The key server broadcasts m encrypted keys as shown below:

$$\text{KeyServer} \longrightarrow * : \{K^0(i-m)\}_{K^0(i-m-1) \oplus K^m(i)}, \dots, \\ \{K^0(i-2)\}_{K^0(i-3) \oplus K^2(i)}, \{K^0(i-1)\}_{K^0(i-2) \oplus K^1(i)}.$$

From step 3, we can see clearly that the ability of a user to recover previous group keys depends on its membership duration. For a new user that only receives $K^0(i)$, it cannot contribute any keys to help any users whose membership was revoked earlier to recover the previous group keys. For a current member that has been in the group for at least m rekeying periods, it can generate all the keys in the key chain after it receives $K^m(i)$; hence it can recover all the m group keys if it has $K^0(i-m-1)$. For a current member that joined at $T(j)$, $i-m < j < i$, it will receive $K^{i-j}(i)$, which enables it to recover at most the keys between $K^0(j)$ and $K^0(i)$ even when it colludes with the early revoked nodes. But this is not an additional security leak, because the user is authorized to have these keys. Thus, this scheme is secure to the rejoining/colluding attack that appears in Scheme I.

An Example In Fig. 6 we show an example to illustrate scheme II. Let $T(10)$ be the current rekeying time and $m = 5$. Following the algorithm above, the key server first generates a random key $K^5(10)$, based on which it generates a hash key chain $K^5(10), K^4(10), \dots, K^1(10), K^0(10)$. The current members of the group are considered to be divided into $m + 1 = 6$ subgroups depending upon their membership duration. The key sent to newly joined members is $K^0(10)$, while the keys sent to the remaining members are as follows: $K^1(10)$ is sent to the users that joined at $T(9)$, $K^2(10)$ is sent to the users that joined at $T(8)$, $K^3(10)$ is sent to the users that joined at $T(7)$, $K^4(10)$ joined at $T(6)$, and $K^5(10)$ to all the users that joined at or before $T(5)$. Finally it broadcasts

$$\text{KeyServer} \longrightarrow * : \{K^0(5)\}_{K^0(4) \oplus K^5(10)}, \{K^0(6)\}_{K^0(5) \oplus K^4(10)}, \dots, \\ \{K^0(8)\}_{K^0(7) \oplus K^2(10)}, \{K^0(9)\}_{K^0(8) \oplus K^1(10)}.$$

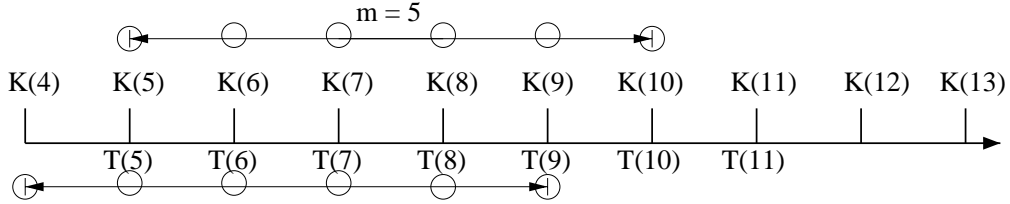


Fig. 6. An example illustrating Scheme II. Here $T(10)$ is the current rekey time.

The Key Distribution Algorithm To evaluate the communication complexity of this scheme, we first discuss the key distribution algorithm that distributes the keys in the key chain to users of different membership durations in step 2 of the scheme. A simple approach for this is to deliver $K^m(i)$ to the current users that joined at or before $T(i-m)$ using the SDR method, while delivering all the keys $K^j(i)$, $m < j \leq 0$, to the users that joined at $T(i-j)$ through unicast, encrypted individually with the leaf keys corresponding to these users. Let $N(i)$ denote the number of users that join the group between $T(i-1)$ and $T(i)$, then the communication complexity of this scheme is $\sum_{j=i-m}^i N(j)$. Thus, the scalability of this approach depends on the arrival rate of the system. Below we present another algorithm that has the communication complexity of $O(m)$.

In SDR, users are mapped into the leaves of the key tree from the left to the right in the same order as their joining times. Hence, all the $N(i)$ users that join between $T(i-1)$ and $T(i)$ and are added into the key tree at $T(i)$ as a batch occupy $N(i)$ consecutive leaves of the key tree. Naturally, they form one or more subsets in the SDR algorithm. The key server uses the following process to determine the number of subsets for these $N(i)$ users:

1. Identifies the minimal full subtree $S(i)$ that covers all these $N(i)$ users. Clearly, the root of this subtree is the least common ancestor of these users.
2. Marks all the leaves in $S(i)$ which are not mapped to any of these $N(i)$ users as revoked.

3. Runs the SDR algorithm to determine the number of subsets for $S(i)$.

The key server maintains a main key tree that corresponds to all the users that joined at least m rekeying periods ago, and m subtrees that correspond to the other users that joined at different rekeying periods. From the m^{th} rekeying on, in every rekeying the key server merges the subtree that corresponds to the users whose membership durations just exceeded m rekeying periods into the main key tree.

As in the original SDR algorithm, the number of subsets in this algorithm increases when users join and leave. The communication complexity of our approach depends upon how many *additional* subsets N_a our algorithm introduces compared to the original SDR algorithm. The value of N_a depends on the group size, the number of arrivals and the number of revoked users in each rekeying period, and the value of m . We studied this issue through extensive simulations. We found that in most cases N_a is smaller than m , whereas in other cases it lies between m and $2m$. Although we do not have an analytical proof for this conclusion, our simulations strongly indicate that the number of additional subsets is less than $2m$. Thus, taking into account the m keys broadcast in step 3 of our scheme, we conclude that we can add the self-healing property to SDR at the expense of transmitting at most $3m$ additional keys.

5 Conclusions

In this paper, we studied two important issues related to the subset difference rekeying method. First, we presented a hybrid key distribution scheme, called WFEC-BKR, that combines the advantages of the proactive FEC and WKA-BKR key delivery protocols. Through simulation, we show that WFEC-BKR has low latency as well as low bandwidth overhead. Second, we present a recovery scheme that adds the self-healing property to SDR with very small additional communication overhead.

References

1. D. Balenson, D. McGrew, and A. Sherman. Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization. IETF Internet draft (work in progress), August 2000.
2. B. Briscoe. MARKS: Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences. In Proc. of First International Workshop on Networked Group Communication, NGC 1999.
3. W.Chen and L.Dondeti. Performance comparison of stateful and stateless group rekeying algorithms. In Proc. of Fourth International Workshop on Networked Group Communication, NGC 2002.
4. D. Naor, M. Naor, and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In Advances in Cryptology - CRYPTO 2001. Springer-Verlag Inc. LNCS 2139, 2001, 41-62.
5. S. Setia, S. Koussih, S. Jajodia. Kronos: A Scalable Group Re-Keying Approach for Secure Multicast. In Proc. of the IEEE Symposium on Security and Privacy, Oakland CA, May 2000.
6. J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin and D. Dean. Self-Healing Key Distribution with Revocation. In Proc. of the IEEE Symposium on Security and Privacy, oakland, CA, May 2002.
7. S. Setia, S. Zhu and S. Jajodia. A Comparative Performance Analysis of Reliable Group Rekey Transport Protocols for Secure Multicast. In Performance Evaluation 49(1/4): 21-41 (2002), special issue Proceedings of Performance 2002, Rome, Italy, Sept 2002.
8. C. Wong, M. Gouda, S. Lam. Secure Group Communication Using Key Graphs. In Proc. of SIGCOMM 1998, Vancouver, British Columbia, 68-79.
9. D. Wallner, E. Harder and R. Agee. Key Management for Multicast: Issues and Architecture. Internet Draft, draft-wallner-key-arch-01.txt, September 1998.
10. Y. Yang, X. Li, X. Zhang and S. Lam. Reliable group rekeying: Design and Performance Analysis. In Proc. of ACM SIGCOMM 2001, San Diego, CA, USA, August 2001, 27-38.
11. S. Zhu, S. Setia, and S. Jajodia. Performance Optimizations for Group Key Management Schemes. In Proc. of the 23rd IEEE ICDCS 2003, Providence, RI, May 2003.