Stateful Subset Cover

Mattias Johansson[1], Gunnar Kreitz[2], and Fredrik Lindholm[1]

[1] Ericsson AB, SE-16480 Stockholm, Sweden
{mattias.a.johansson, fredrik.lindholm}@ericsson.com
[2] Royal Institute of Technology, Stockholm, Sweden
gkreitz@nada.kth.se

Abstract. This paper describes a method to convert stateless key revocation schemes based on the subset cover principle into stateful schemes. The main motivation is to reduce the bandwidth overhead to make broadcast encryption schemes more practical in network environments with limited bandwidth resources, such as cellular networks. This modification is not fully collusion-resistant.

A concrete new scheme based on the Subset Difference scheme [1] is presented, accomplishing a bandwidth overhead of $\Delta m + 2\Delta r + 1$ compared to e.g. Logical Key Hierarchy's $2(\Delta m + \Delta r)\log m$, where Δm and Δr is the number of members added and removed since the last stateful update and m is the number of current members.

Keywords: Broadcast encryption, key revocation, subset cover, Subset Difference, Logical Key Hierarchy, stateful, stateless.

1 Introduction

In this paper we show how a key server can establish a common group key K_g for a dynamically changing group (i.e., members can join and leave). One possible application area is the protection of broadcast streams (e.g., internet or mobile broadcasting of movies, music, or news), and the topic is therefore generally referred to as broadcast encryption. The group key which is to be distributed is often referred to as the media key, or session key.

This problem is well studied and is usually solved by using a key revocation scheme. One large class of key revocation schemes are the subset cover schemes, introduced in [1]. In this paper we present a general method for adding state to subset cover schemes, which reduces the bandwidth overhead greatly.

In the system setup stage, the key server gives each user u some key information K_u. This information can be thought of as a set of keys; in general it will be information from which keys can be derived. The size of K_u is called the *user storage*. Schemes where K_u is never updated are called *stateless*, whereas those where it is updated K_u are called *stateful*.

Every time a new group key is distributed, the key server will broadcast a *header*, using which all legitimate group members can calculate the new group key. The size of this header is called the *bandwidth overhead*, and the time it takes for a member to compute the group key from the header and her set of key information K_u is called the *computational overhead*.

1.1 Preliminaries

Broadcast encryption was first introduced by Berkovits in [2], and later Fiat and Naor started a more formal study of the subject [3]. The first practical broadcast encryption scheme was the stateful Logical Key Hierarchy (LKH) scheme proposed in [4, 5]. LKH accomplishes a worst case bandwidth overhead of $2(\Delta m + \Delta r) \log m$, where Δm and Δr are the number of added and removed members since the last stateful key update, and m is the number of current members.

Later, the class of schemes known as subset cover schemes, and the Subset Difference (SD) scheme were presented in [1]. Further variants of the SD scheme have been developed in [6, 7]. Other subset cover schemes include the Hierarchical Key Tree scheme [8], and the Punctured Interval scheme, π [9]. All of these schemes have bandwidth overhead which is linear in r.

Stateful or Stateless. The advantage of a stateless scheme compared to a stateful scheme is that a member does not have to receive all previous updates in order to decrypt the current broadcast. In many settings, this advantage is not as big as it first appears. Stateful schemes can be augmented with reliable multicast techniques or can make missed broadcasts available on request. Also, in e.g. a commercial settings where the group key is updated every five minutes, a stateless scheme would also need to use similar techniques, since missing five minutes of content due to a single packet lost would be unacceptable.

Notation. Let $\mathcal{M}$ be the set of members of the group, $\mathcal{R}$ be the set of revoked users and $\mathcal{U}$ be the total set of users, or potential members (i.e. the union of $\mathcal{M}$ and $\mathcal{R}$). Let m, r and u be the sizes of these sets. Let Δm and Δr be the number of users who have joined and left the group since the last *stateful* update. Let $E_K(M)$ be the encryption of message M under key K. In a binary tree, let $\mathrm{l}(v)$ and $\mathrm{r}(v)$ be the left and right child of node v. Let $\mathrm{par}(v)$ and $\mathrm{sib}(v)$ be the parent and sibling of node v.

1.2 Our Contribution

Subset cover schemes define a family of subsets of $\mathcal{U}$, where each subset is associated with a key. To distribute a new group key, the key server covers $\mathcal{M}$ (and avoids $\mathcal{R}$) with subsets from the family and encrypts the new group key K_g using the key of each subset used in the cover. We present a technique where a *state key*, K_S is added, which is held by current members of the group.

When distributing a new group key, the state key is used to transform all subset keys. Since only current members have access to the state key, the key server does not need to avoid covering all of $\mathcal{R}$, but only those who were recently removed (and thus have a current state key).

This technique can be applied to any scheme based on the subset cover principle. It is often beneficial to develop a new algorithm to calculate the cover, and this has been done for the SD scheme.

1.3 Organization of This Paper

In Section 2, a brief overview of subset cover schemes is given. In Section 3, our idea, Stateful Subset Cover, is presented in detail. In Section 4, we show practical performance results on some simulated datasets. In Section 5, we discuss the security of our proposed scheme. We give concluding remarks in Section 6. Algorithms for Stateful Subset Cover are given in Appendix A.

2 Subset Cover Schemes

A general class of stateless schemes are called subset cover schemes and were first introduced in [1]. In this class of revocation schemes, there is a preconfigured family of sets, $\mathcal{F} = \{f_1, f_2, \ldots\}, f_i \subseteq \mathcal{U}$. Each set $f_i \in \mathcal{F}$ has an associated key K_i such that each user belonging to f_i can compute K_i, but no user outside of f_i can compute K_i.

To distribute a new group key, the key server calculates an exact cover $\mathcal{F}'$ of $\mathcal{M}$, i.e. $\mathcal{F}' = \{f_{i_1}, f_{i_2}, \ldots\} \subseteq \mathcal{F}$ and $\bigcup \mathcal{F}' = f_{i_1} \cup f_{i_2} \cup \ldots = \mathcal{M}$. The key server then broadcasts the following message:

$$\mathcal{F}', \mathrm{E}_{K_{i_1}}(K_g), \mathrm{E}_{K_{i_2}}(K_g), \ldots$$

where $\mathcal{F}'$ here denotes some suitable representation of $\mathcal{F}'$ such that members can compute what part of the message to decrypt using what key. Since the sets $\mathcal{F}$ and the keys associated with the sets are fixed, this broadcast encryption scheme is stateless.

2.1 Subset Difference

In the Subset Difference (SD) scheme, which is a subset cover scheme, every user is associated with a leaf in a binary tree. For every node v in the tree, and every node w below v, we have $S_{v,w} \in \mathcal{F}$, where $S_{v,w}$ is the set of all leaves in the subtree rooted in v, except for those in the subtree rooted in w. In figure 1, two such sets are shown, the set $S_{2,10}$ and the set $S_{6,12}$. The corresponding broadcast from the key server would in this case be

$$\{S_{2,10}, S_{6,12}\}, E_{K_{S_{2,10}}}(K_g), E_{K_{S_{6,12}}}(K_g).$$

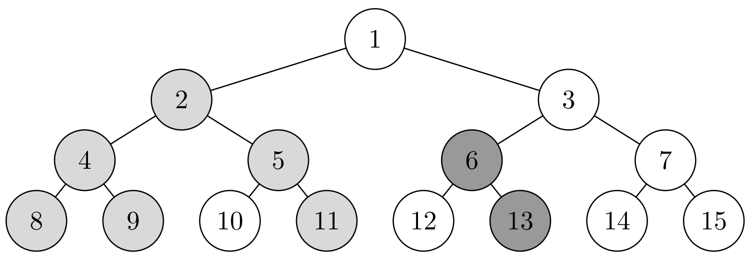

Fig. 1. The sets $S_{2,10}$ (light) and $S_{6,12}$ (dark) in an SD tree

A user is not given the keys $K_{S_{v,w}}$ she is entitled to directly, since that would consume too much user memory. Instead, she is given $\mathcal{O}(\log^2 u)$ values from which the keys she should have access to can be derived by $\mathcal{O}(\log u)$ applications of a pseudo-random number generator. For details on how the key derivation in SD works, see [1]. The SD scheme has a bandwidth overhead which is $\min(2r + 1, m)$.

2.2 The Punctured Interval Scheme π

In the punctured interval (π) scheme, users can be thought of as being on a line, each user indexed by an integer. The subsets used in the π scheme are of the form $S_{i,j;x_1,\ldots,x_q} = \{x | i \leq x \leq j, x \neq x_k, 1 \leq k \leq q\}$, i.e. all users between positions i and j (inclusive) except for the q users $x_1, \ldots, x_q$.

The scheme has two parameters, p and c affecting the performance of the system. The parameter c is the maximum length of the interval, e.g. $1 \leq j - i + 1 \leq c$, and the parameter p is how many users in the interval can at most be excluded, e.g. $0 \leq q \leq p$. Large p and c lower bandwidth requirements but increase user storage and computational overhead. The bandwidth overhead is about $\frac{r}{p+1} + \frac{u-r}{c}$ and the user storage is $\mathcal{O}(c^{p+1})$. For details on the π scheme, see [9].

3 Stateful Subset Cover

In this section, a general technique for transforming a subset cover scheme into a stateful scheme through the introduction of a *state key* is presented. This makes the bandwidth performance linear in $\Delta m + \Delta r$ instead of in r. As will be discussed further in Section 5, this weakens the security of the system somewhat in that it opens up the opportunity for collaboration. This risk can however be mitigated by periodically using the normal update mechanism of the underlying subset cover scheme, which is referred to as a *hard update*.

3.1 An Intuitive Description

Recall that, as presented in Section 2, a subset cover scheme by covering members using a static family of subsets of users. The subsets are created at startup-time and are constant throughout the life of the system. Each subset is associated with a key that only users in that subset have access to. To distribute a new group key, the key server broadcasts the new group key encrypted with the key for each subset in the cover.

We introduce a general extension to a subset cover scheme by adding a state key, which is distributed alongside with the group key to members. This state key is then used for distributing the next group key and state key. When the key server broadcasts a new group key (and state key) it will not encrypt the new group key directly with the keys of the selected subsets, instead it will use the key of the selected subset transformed by the current state key using some suitable function (e.g. xor). This means that to decrypt the new group key, a user must not only be in a selected subset, but must also have the current state key.

The key server, when it saves bandwidth doing so, is thus free to cover revoked users too, as long as they do not have access to the current state key. So, the key server need only avoid covering those who were revoked recently. To discourage cheating (see Section 5) the aim is to cover as few revoked users as possible, which is referred to as a *cheap* cover.

The alert reader may have noticed a problem with the system as described. If the state key is needed to decrypt the new group key and state key, how are recently joined members, who do not have the current state key, handled? The answer is that the state key is not used when covering joiners, and thus all of $\mathcal{R}$ must be avoided. However, the scheme is free to cover current members who have a state key, and it is preferable for it to cover as many of these as it can, since those covered here will not need to be covered in the cover using the state key. This is called a *generous* cover.

A variation of the above extension is to run the system in a *semi-stateless* mode. That means that the key server in each round is free to decide whether it wishes to update the state key or not. As long as the state key is not updated, the scheme will have the properties of a stateless scheme, but the bandwidth usage will gradually increase since Δm and Δr (membership changes since last *stateful* update, see Section 1.1) will increase. A group key update when the state key is changed is called a *stateful update* and one where the state key remains unchanged is called *stateless update.*

3.2 Generalized Stateful Subset Cover

For this type of scheme to work, a new cover function is needed. The traditional subset cover has two types of users: members and revoked, or blue and red. The new cover function has three types of users: must cover (MC), can cover (CC) and must not cover (NC). The output is a cover covering all users marked MC and not covering any NC users. Users marked as CC, can be either covered or not covered.

As discussed in Section 3.1, there are two versions of the cover algorithm for each scheme, *generous* and *cheap.* Both versions will primarily minimize the number of subsets used for the cover. The generous cover will attempt to cover as many CC users as possible and the cheap cover will cover as few as possible.

More formally, we have a new decision problem, Optional-Set-Cover($\mathcal{F}$, $\mathcal{M}$, $\mathcal{R}$, k, n), where $\mathcal{F}$ is a family of subsets of some finite set $\mathcal{U}$, $\mathcal{M}$ is subset of the same $\mathcal{U}$ and k and n are integers. The problem is: is there a subset $\mathcal{F}' \subseteq \mathcal{F}$ such that $\bigcup \mathcal{F}' \supseteq \mathcal{M}$, $|\mathcal{F}'| = n$, $|\bigcup \mathcal{F}'| = k$, and $(\bigcup \mathcal{F}') \cap \mathcal{R} = \emptyset$?

The two optimization problems, generous and cheap, both primarily want to minimize n, and then on the second hand either want to maximize or minimize k, respectively. The optimization problems are denoted Generous-Cover($\mathcal{F}$, $\mathcal{M}$, $\mathcal{R}$) and Cheap-Cover($\mathcal{F}$, $\mathcal{M}$, $\mathcal{R}$). The (optional) subset cover problem is in the general case NP complete, but subset cover schemes are designed in such a way that an efficient algorithm exists.

The Framework. The system is initialized exactly as the underlying subset cover scheme, with one exception. A state key, K_S, is generated by the key server and given to all initial members of the system. The key server also keeps track of the set of users to which it has given the current state key, the set $\mathcal{S}$.

To update the group key, the key server first decides whether it is time to do a hard update or not. If a hard update is done, it uses the underlying scheme to distribute a new K'_g and K'_S and sets $\mathcal{S} \leftarrow \mathcal{M}$.

If it was not time to do a hard update, it begins by calculating a cover $\mathcal{C}_1$ as $\mathcal{C}_1 \leftarrow \textsc{Generous-Cover}(\mathcal{F}, \mathcal{M}\backslash\mathcal{S}, \mathcal{R})$. Note that this cover is empty if $\mathcal{M}\backslash\mathcal{S}$ is empty, i.e. if no new members have been added since K_S was last updated.

After this, it checks if $\mathcal{R} \cap \mathcal{S} = \emptyset$. If this is the case, i.e. no one has been removed since K_S was last updated, no one besides members has K_S, and thus K_S can be used to securely communicate with members. If $\mathcal{R} \cap \mathcal{S} \neq \emptyset$, it instead calculates $\mathcal{C}_2 \leftarrow \textsc{Cheap-Cover}(\mathcal{F}, \mathcal{M}\backslash(\bigcup \mathcal{C}_1), \mathcal{R} \cap \mathcal{S})$.

The key server then first broadcasts a description of the covers $\mathcal{C}_1$, $\mathcal{C}_2$ in some form, so that members know what part of the broadcast to decrypt. The message will then consist of, firstly, for every $c \in \mathcal{C}_1$: $\mathrm{E}_{K_c}(K_E)$, where K_c is the key associated with subset c. Secondly, if $\mathcal{R} \cap \mathcal{S} \neq \emptyset$, the message will contain, for every $c \in \mathcal{C}_2$: $\mathrm{E}_{K_c}(K_F)$, or if $\mathcal{R} \cap \mathcal{S} = \emptyset$, $\mathrm{E}_{K_S}(K_F)$. We let $K_E = f(K_F, K_S)$, where f is a suitable function, such as xor. Finally, the message will contain $\mathrm{E}_{K_E}(K'_g, K_S)$ or $\mathrm{E}_{K_E}(K'_g, K'_S)$ depending on if it was a stateless or stateful update, respectively. If the update was stateful, the key server sets $\mathcal{S} \leftarrow \mathcal{M}$, otherwise $\mathcal{S}$ is left unchanged.

Generic Cover. A normal subset cover algorithm can be used to solve the optional subset cover problem, but generally not optimally. Since most subset cover schemes have bandwidth performance which linear in r, it is often beneficial to minimize $\mathcal{R}$.

So, for the optional subset cover problem, we can simply re-mark all CC users as members and then run the normal subset cover algorithm on the resulting set. Post-processing can be done to remove any sets covering only users who were labelled CC before the re-marking. Post-processing can also, in the cheap variant, attempt to narrow a set down (i.e. change the set so that fewer CC users are covered).

For a specific underlying scheme, it is often possible to make better use of the CC users than this rather naïve transformation. An optimal algorithm for the SD scheme will be discussed in the next section.

3.3 Stateful Subset Difference

For SD (Section 2.1), which is one of the most important subset cover schemes, a new cover algorithm has been developed. Pseudo-code for the algorithm can be found in Appendix A, but we describe and discuss it here.

The complexity of the algorithm has the same asymptotic complexity as the original. This algorithm could also be used in stateful variants of other subset cover schemes which use the SD cover method, such as LSD and SSD ([6, 7]).

Let each node v have three variables, two booleans, v**.mc** and v**.nc** and one integer, v**.cc**. The variable v**.cc** counts the number of CC users which can be excluded under v. If v**.mc** is true, it means that there are uncovered MC users (i.e., users which must be covered) below v. If v**.nc** is true, it means that there are NC users (i.e., users which must not be covered) below v.

A basic observation is that a node v where both the left ($\mathrm{l}(v)$) and the right child ($\mathrm{r}(v)$) has a NC node below it cannot be used as a top node for a key. So, if any MC nodes are below such a node, the top node to use for the set cannot be higher up in the tree than $\mathrm{l}(v)$ or $\mathrm{r}(v)$.

For the algorithms given, we assume that the bottom nodes (i.e., the user nodes) have already been colored in the input. That means that for a MC user at node v, v**.mc** = **true**, v**.nc** = **false** and v**.cc** = 0. Analogously for a NC user. For a CC user at node v, v**.mc** = v**.nc** = **false** and v**.cc** = 1.

The algorithm consists of three functions. Cover() is the top-level function which is called to generate the entire cover, with a parameter telling it if a generous or cheap cover is wanted. Cover() "adds up" the marks of the child nodes, and call a helper-function to calculate the exact subset when it discovers that a subset has to be placed. When subsets are placed, they are guaranteed to cover all remaining MC users under the current node. This means that there is nothing left to be covered below that node, so the parent will be marked with only NC, instead of the usual "sum" of the child nodes.

The two functions, Generous-Find-Subset() and Cheap-Find-Subset() calculate a single subset, given a top node which is the highest place for the top node of the subset. Both versions use the markings placed by Cover() to calculate the subset. The generous version will just ensure that no NC users are covered, while the cheap version will both ensure that no NC users are covered and will attempt to exclude as many CC users as possible.

As an example, consider the tree shown in Figure 2. Let the letter 'N' denote v**.nc** = **true**, 'M' denote v**.mc** = **true** and 'C' denote v**.cc** = 1. The bottom nodes have all been colored in the input to the algorithm. Going through the nodes in depth-first (left-to-right) order, in the first node, the NC and CC marks from the child nodes are combined into the parent. In its sibling, the CC and NC marks propagate up.

When we get to the left child of the root, then both children are marked with NC and at least one child is marked with MC. This will cause Find-Subset() to be called. In both cases, Find-Subset() will select the subset $S_{4,8}$, which covers the single MC node 9.

For the right subtree, marks will propagate upwards and the coloring will reach the right child of the root without Find-Subset() being called. However, both children of the root are marked NC, and node 3 is marked MC, so one of the Find-Subset() functions will be called again. In this case, the subset $S_{3,14}$ will be selected, which covers the two MC users 12 and 14 and the CC user 13.

This algorithm differs somewhat from the original cover algorithm for SD given in [1]. The original algorithm begins by calculating the Steiner tree of the revoked users and the root and then calculate the cover directly from the

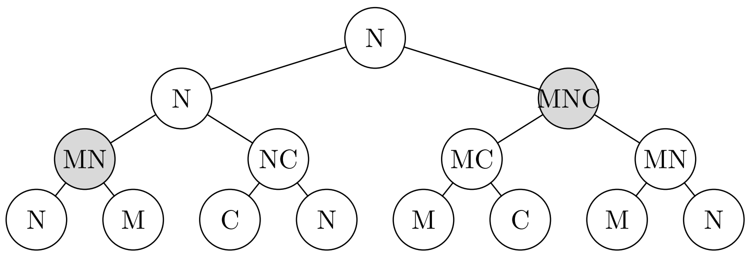

Fig. 2. A sample coloring of a stateful subset difference tree, gray nodes show on what nodes FIND-SUBSET() are called

properties of the Steiner tree. The algorithm presented here also works as a cover algorithm for the normal SD cover problem. Both algorithms have the same time complexity.

3.4 Stateful Punctured Interval

A very simple greedy (and suboptimal) cover algorithm has been tested with the π scheme. We do not present this algorithm here. Recall (Section 2.2) that this scheme has two parameters which can be tuned. Tests have been performed with four sets of parameters, $(c, p) = (1000, 1), (100, 2), (33, 3), (16, 4)$. These were selected such that user storage would be approximately 1 Mbyte, which we deemed reasonable for many scenarios. We present the results parameters giving the best results on our dataset, $(c, p) = (1000, 1)$. Further tuning may give even better performance.

3.5 Performance

The user storage will essentially be unchanged (a single extra key needs to be stored by members and the key server) by the addition of state, so they will be the same as those of the underlying scheme. Analogously for the computational overhead. The scheme will take on the negative properties of a stateful scheme in that packet loss becomes a more serious issue which will need to be handled, see the discussion in Section 1.1, where we argue that this is not as big a drawback as it first appears.

Bandwidth Impact. The bandwidth performance will in general improve. The bandwidth usage for the first cover calculated (for joining members, where the state key is not used) is at worst that of the underlying scheme with Δm members and r revoked users. For the second cover, where the state key is used, the performance is at worst that of the underlying scheme with m members and Δr revoked users. Inserting the values for SD, we get a worst-case bandwidth performance of $\min(2r+1, \Delta m)+\min(2\Delta r+1, m)$, which will, in most situations, be $\Delta m + 2\Delta r + 1$. This can be compared to for instance LKH, which has a performance of $2(\Delta m + \Delta r) \log m$.

The worst-case performance is better than previous protocols. A comparative performance evaluation in several usage scenarios has also been performed, and some of these results are presented in Section 4.

Computational Complexity for General Stateful Subset Cover. At most u users can be undecided, so at worst, u nodes must be re-labeled. The output of the cover will cover each undecided or cover node exactly once, so the cost of going through the sets is at most the number of such nodes, which is u. Thus, the added runtime for both the pre-processing and post-processing steps is $\mathcal{O}(u)$.

Computational Complexity for Stateful Subset Difference. The performance of the cover algorithm is $\mathcal{O}(u)$. On the way up, each node will be visited exactly once and the tree has $\mathcal{O}(u)$ nodes. When FIND-SUBSET() is called, the top node will only be colored red. All downward traversal in the FIND-SUBSET() functions will always stop at a node colored only red. This means that on the way down (i.e., in FIND-SUBSET()) each node can be visited at most twice.

3.6 Correctness

The framework is correct in that a member can recover the new key, as long as she has not missed the last stateful update. For a member there are two cases. Either, she is recently added and does not have the current state key, or she has the current state key.

If a member m does not have the current state key, then $\mathcal{M} \backslash \mathcal{S} \neq \emptyset$ since it must at least contain m. If so, the cover $\mathcal{C}_1 = \text{GENEROUS-COVER}(\mathcal{M} \backslash \mathcal{S}, \mathcal{R})$ will be calculated and the underlying scheme will be used to distribute the keys with the resulting cover. If the underlying scheme is correct, the member will be able to recover the key and the current state key.

If, on the other hand, m does have the current state key, there are three cases. If m is covered by $\mathcal{C}_1$ then she can discover that fact by looking at the information about the cover and recover the key, given that the underlying scheme was correct.

If $\mathcal{R} \cap \mathcal{S} = \emptyset$, then the new group key will be distributed as $\mathrm{E}_{K_S}(K'_g)$ and since m has K_S, she can recover the new group key.

If $\mathcal{R} \cap \mathcal{S} \neq \emptyset$, then a second cover, $\mathcal{C}_2 = \text{CHEAP-COVER}(\mathcal{M} \backslash (\bigcup \mathcal{C}_1), \mathcal{S} \cap \mathcal{R})$ will be calculated. Since m has the state key and was not covered by $\mathcal{C}_1$, she will be covered by $\mathcal{C}_2$. Given that the underlying scheme is correct, she can then recover K_F, from which she can derive K_E since $K_E = f(K_F, K_S)$. She can then recover the new group key which is distributed as $\mathrm{E}_{K_E}(K'_g)$.

4 Practical Results

For simulation purposes, two datasets have been used. In the first dataset, the number of users currently in the group follow a sinus-shaped form, and in the second dataset, the number of users go through (almost) the entire range of the system, from all users being members to almost all users being revoked.

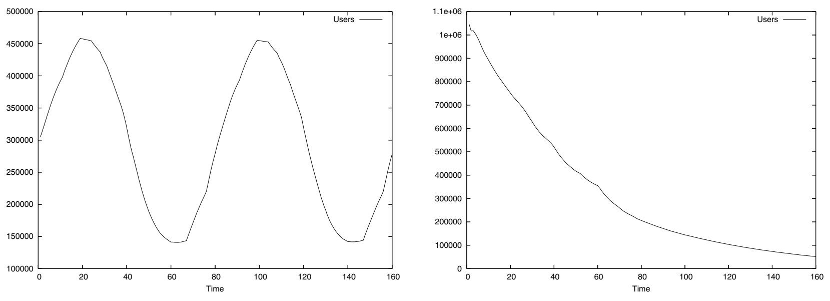

Fig. 3. The sinus-shaped and full-ranged dataset with 2^{20} users

Key updates occur only at discrete intervals (i.e., in batch mode), of which there were 160 in the simulation. To increase the dynamic of the system, except for the joins and leaves necessary to generate the proper form, a *base change* rate was added. The base change is a value between 0 and 1 signifying how large fraction of members that will be replaced by non-members during each round. Given that the scheme we present performs worse in a highly dynamic system, we have used simulations with a very high basechange of 2% to show that it still performs well under difficult conditions. The datasets are displayed in Figure 3. More simulation results are in [10].

The performance of the stateful subset cover schemes presented in this article were evaluated and compared to the performance of the popular LKH scheme, as well as the stateless subset schemes. Note that the stateless schemes do have significantly different performance characteristics, and will e.g. behave poorly when a majority of the population is revoked.

4.1 Performance in Stateful Subset Cover

The performance of the stateful SD and π schemes was evaluated using the scenarios presented in the previous section. As will be shown, the performance is significantly better than that of the LKH scheme, as could be expected from the theoretical analysis.

Figure 4 shows a comparison between two variations of stateful subset cover and LKH. The regular, stateless versions of the subset cover schemes were omitted from this figure for clarity. They do, in fact, have better performance than LKH in this scenario (due to the high base change rate), but the stateful variations still significantly outperform them.

Table 1 shows both the average number of sets used per key update and the maximum number of sets used for a single update. Both these measurements are important to minimize. Minimizing the average will keep total bandwidth usage down and minimizing the maximum will keep the latency for key refresh reasonably low. We show results without any hard updates in these tables. With periodic hard updates, the maximum for the stateful versions will be (about) the same as for the normal versions of the schemes.

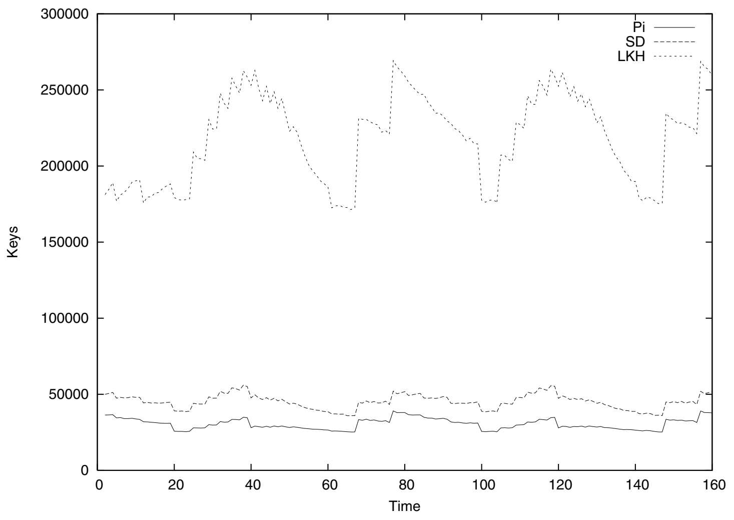

Fig. 4. Bandwidth performance in the sinus-shaped dataset

Table 1. Performance comparison between normal and stateful subset cover schemes (without hard updates), and LKH

	Sinus dataset		Full-range dataset	
Scheme	Avg. sets used	Max sets used	Avg. sets used	Max sets used
Stateful SD	45 136	55 983	43 214	59 798
Stateful π	30 549	39 067	28 153	33 980
LKH	218 046	269 284	241 362	393 758
Normal SD	222 409	295 778	170 175	305 225
Normal π	153 733	180 157	114 100	180 048

As the diagram and table shows, the stateful version acheives significantly lower bandwidth requirements compared to previous schemes. The worst-case performance (i.e., the maximum number of sets used) is reduced by a factor five and the average case is reduced by a factor four.

5 Security of Stateful Subset Cover

The scheme as presented is not fully collusion-resistant. Users can collaborate by one revoked user who has recently been removed sharing the current state key with a user who is covered, but who does not have the state key. In this section, techniques for mitigating this type of attacks will be discussed. Further, a model for this type of cheating will be given and, the expectancy for how long cheaters gain access will be analyzed using the same data as was used for performance evaluations.

5.1 Security Model

In a commercial setting, the concern is to make it cost-inefficient rather than impossible to illegally decrypt the broadcast. In this model it is assumed, that it is possible to make it expensive to extract state keys, by putting them in a protected area such as a smart card or other tamper resistant hardware. In addition, by using periodic hard updates, cheaters will also periodically be removed from the system. These two techniques can together be used to mitigate the effect of a collaborative behavior by dishonest users.

The major threat in a commercial setting would be the extraction of legitimate long-term keys, since that would allow for pirate decoders to be manufactured. This can be made hard by placing the long-term keys in protected hardware, and by using *traitor tracing* techniques, should the keys leak. Concerning this threat, the stateful subset cover schemes presented here have the same security properties as regular subset cover schemes, given that the long-term key structure is identical.

Another important threat would be redistribution of the group key by a member, i.e. every time a new group key is distributed, the dishonest (but paying) member sends the new group key to her friends. The group key is identical for all users in the group, so traitor tracing techniques are not applicable.

In the stateful schemes, another option would be for a dishonest user to instead redistribute the state key (along with the group key) to her friends. The advantage to the cheaters would be that this would not have to be done as frequently as the group key is to be distributed.

An important aspect to analyze is the expectancy of the time a user who illegally gets a state key can recover the group key. In our model, every user who is removed from the system is given the next round's state key and group key, so she can recover the group key for at least one more round. Then, as long as she is covered using the state key, she can keep decrypting, but as soon as she is not covered in a round, she will lose her ability to decrypt the broadcast. The larger this expectancy, the more seldom a traitor would need to redistribute the state key to keep enabling her friends to recover the group key.

This model of cheating simulates a user illegally receiving a state key. It is run over the same simulation data as the performance tests to give a real-world like cover. This also means that some users will be added again before they are successfully revoked by the system. These are ignored when calculating the average.

In the simulation, we measure the average number of rounds users who left in round r could watch the show, given that they were given one key. In the simulations, a hard update is always run immediately after the end of the simulation, i.e. it is assumed that after the last round, all current cheaters were revoked.

5.2 Subset Difference

In Figure 5, the average numbers of rounds a revoked user can watch if she receives that key is shown. The cheating model used is described in more detail in the previous section. As can be seen, even with hard updates done every 160:th round, a cheater can still at best expect to see approximately 10 rounds.

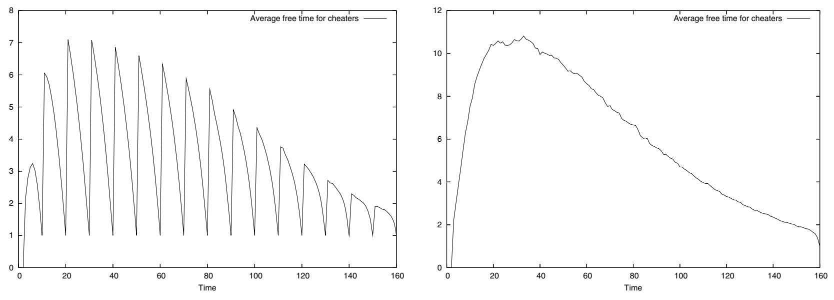

Fig. 5. Average free time for cheaters in stateful SD. 2^{20} users, full-range dataset, basechange 2%, hard updates every 10 (left) and 160 (right) rounds.

While this is worse than the normal schemes, where this number is constantly 1, it is still reasonably small.

At the cost of bandwidth, the frequency with which a traitor must redistribute the state key can be increased by doing hard updates more frequently.

6 Summary

This paper introduces the idea of adding state to a certain class of key revocation schemes, called subset cover schemes. Having state in a key revocation scheme has some drawbacks, like an increased vulnerability to packet loss. These drawbacks are not as bad as they first appear, as we argue for in Section 1.1.

The specific method used in this paper is not collusion-resistant by itself, but may need additional mitigation techniques, such as tamper resistant modules for acceptable security. This non-perfect security is however by practical examples shown to have a limited effect on the overall security from a commercial point of view, where the interest is more directed towards making illegal decryption cost-inefficient rather than impossible.

As a benefit, it is shown that the conversion of a stateless subset cover scheme may lead to greatly reduced bandwidth overhead. This is extremely important in network environments where the available bandwidth is a limited resource, like for example cellular networks. In particular, simulation results show a significant reduction of bandwidth compared to previous schemes.

The transformation presented in this paper is not very complex, with the addition of a global state key, common to all members. Could there be more advanced transformations of subset cover (or other broadcast encryption) schemes which further reduce bandwidth overhead or give better security properties?

References

1. D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers," *Lecture Notes in Computer Science*, vol. 2139, pp. 41–62, 2001.
2. S. Berkovits, "How to broadcast a secret," *Lecture Notes in Computer Science*, vol. 547, pp. 535–541, 1991.

3. A. Fiat and M. Naor, “Broadcast encryption,” *Lecture Notes in Computer Science*, vol. 773, pp. 480–491, 1994.
4. D. M. Wallner, E. J. Harder, and R. C. Agee, “Key management for multicast: Issues and architectures,” Internet Request for Comment RFC 2627, Internet Engineering Task Force, 1999.
5. C. K. Wong, M. Gouda, and S. S. Lam, “Secure group communications using key graphs,” in *Proceedings of the ACM SIGCOMM ’98 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 68–79, ACM Press, 1998.
6. D. Halevy and A. Shamir, “The LSD broadcast encryption scheme,” *Lecture Notes in Computer Science*, vol. 2442, pp. 47–60, 2002.
7. M. T. Goodrich, J. Z. Sun, and R. Tamassia, “Efficient tree-based revocation in groups of low-state devices,” *Lecture Notes in Computer Science*, vol. 3152, pp. 511–527, 2004.
8. T. Asano, “A revocation scheme with minimal storage at receivers,” *Lecture Notes in Computer Science*, vol. 2501, pp. 433–450, 2002.
9. N.-S. Jho, J. Y. Hwang, J. H. Cheon, . M.-H. Kim, D. H. Lee, and E. S. Yoo, “One-way chain based broadcast encryption schemes,” *Lecture Notes in Computer Science*, vol. 3494, pp. 559–574, 2005.
10. G. Kreitz, “Optimization of broadcast encryption schemes,” Master’s thesis, KTH – Royal Institute of Technology, 2005.

A Algorithms

CHEAP-FIND-SUBSET(T, v)
Input: T is a stateful SD tree where all nodes up to v have been marked by FIND-COVER()
Output: A subset with top node in v or below, covering all uncovered MC users below v, and as few CC users as possible.

```
(1)   if not $v$.nc and not $v$.cc
(2)       if $v = \text{root}$
(3)           return $S_{1,\Phi}$ //(all users)
(4)       else
(5)           return $S_{\text{par}(v),\text{sib}(v)}$
(6)   if not $\text{l}(v)$.mc
(7)       return CHEAP-FIND-SUBSET($T$, $\text{r}(v)$)
(8)   else if not $\text{r}(v)$.mc
(9)       return CHEAP-FIND-SUBSET($T$, $\text{l}(v)$)
(10)  $excl \leftarrow v$
(11)  while $excl$.mc
(12)      if $\text{l}(excl)$.nc
(13)          $excl \leftarrow \text{l}(excl)$
(14)      else if $\text{r}(excl)$.nc
(15)          $excl \leftarrow \text{r}(excl)$
(16)      else if $\text{l}(excl)$.cc $>$ $\text{r}(excl)$.cc
(17)          $excl \leftarrow \text{l}(excl)$
(18)      else
(19)          $excl \leftarrow \text{r}(excl)$
(20)  return $S_{v,excl}$
```

GENEROUS-FIND-SUBSET(T, v)
Input: T is a stateful SD tree where all nodes up to v have been marked by FIND-COVER()
Output: A subset with top node in v or below, covering all uncovered MC users below v, and as many CC users as possible.

```
(1)   if not $v$.nc
(2)       if $v = \text{root}$
(3)           return $S_{1,\Phi}$ //(all users)
(4)       else
(5)           return $S_{\text{par}(v),\text{sib}(v)}$
(6)   $excl \leftarrow n$
(7)   while not $\text{l}(excl)$.nc or not $\text{r}(excl)$.nc
(8)       if $\text{l}(excl)$.nc
(9)           $excl \leftarrow \text{l}(excl)$
(10)      else
(11)          $excl \leftarrow \text{r}(excl)$
(12)  return $S_{v,excl}$
```

```
FIND-SUBSET($T$, $v$, $generous$)
Input: $T$ is a stateful SD tree where all nodes up to $v$ have been marked by FIND-COVER()
Output: A generous or cheap subset with top node in $v$ or below, covering all uncovered MC users below $v$.
(1)   if $generous = \mathbf{true}$
(2)       return GENEROUS-FIND-SUBSET($T$, $v$)
(3)   else
(4)       return CHEAP-FIND-SUBSET($T$, $v$)
```

```
FIND-COVER($T$, $generous$)
Input: $T$ is a stateful SD tree where the nodes representing users have been marked.
$generous$ is a boolean, true for generous cover, false for cheap
Output: A cover $\mathcal{C}$
(1)   $\mathcal{C} \leftarrow \emptyset$
(2)   foreach node $v \in T$ in depth-first order
(3)       if $\mathrm{l}(v)$.nc and $\mathrm{r}(v)$.nc and ($\mathrm{l}(v)$.mc or $\mathrm{r}(v)$.mc)
(4)           if $\mathrm{r}(v)$.mc
(5)               $\mathcal{C} \leftarrow \mathcal{C} \cup$ FIND-SUBSET($T, \mathrm{r}(v), generous$)
(6)           if $\mathrm{l}(v)$.mc
(7)               $\mathcal{C} \leftarrow \mathcal{C} \cup$ FIND-SUBSET($T, \mathrm{l}(v), generous$)
(8)           $v$.nc $\leftarrow$ true
(9)       else
(10)          $v$.nc $\leftarrow \mathrm{l}(v)$.nc | $\mathrm{r}(v)$.nc
(11)          $v$.mc $\leftarrow \mathrm{l}(v)$.mc | $\mathrm{r}(v)$.mc
(12)          if $v$.mc
(13)              $v$.cc $\leftarrow \max(\mathrm{l}(v)$.cc$, \mathrm{r}(v)$.cc$)$
(14)          else
(15)              $v$.cc $\leftarrow \mathrm{l}(v)$.cc $+$ $\mathrm{r}(v)$.cc
(16)  if root.mc
(17)      $\mathcal{C} \leftarrow \mathcal{C} \cup$ FIND-SUBSET($T$, root, $generous$)
(18)  return $\mathcal{C}$
```