Efficient State Updates for Key Management

Benny Pinkas

HP Labs
Princeton, NJ, USA
benny.pinkas@hp.com

Abstract

Encryption is widely used to enforce usage rules for digital content. In many scenarios content is encrypted using a group key which is known to a group of users that are allowed to use the content. When users leave or join the group the group key must be changed. The LKH (Logical Key Hierarchy) algorithm is a very common method of managing these key changes. In this algorithm every user keeps a personal key composed of $\log n$ keys (for a group of n users). A key update message consists of $O(\log n)$ keys.

A major drawback of the LKH algorithm is that users must update their state whenever users join or leave the group. When such an event happens a key update message is sent to all users. A user who is offline during t key updates, and which needs to learn the keys sent in these updates as well as update its personal key, should receive and process the t key update messages, of total length $O(t \log n)$ keys. In this paper we show how to reduce this overhead to a message of $O(\log t)$ keys. We also note that one of the methods that are used in this work to reduce the size of the update message can be used is other scenarios as well. It enables to generate n pseudo-random keys of length k bits each, such that any *successive* set of t keys can be represented by a string $\log(t) \cdot k$ bits, without disclosing any information about the other keys.

1 Introduction and Motivation

Digital Rights Management (DRM) systems provide content which is accompanied by rules or controls that define the ways in which the content can be used. The rules are enforced by a governance mechanism that ensures that only legitimate operations can be applied to the content. The most simple governance mechanism is encryption: The content is encrypted and the decryption key is only given to users which are allowed to use the content[1].

To model this setting we consider the following simplified scenario. A group U of n parties is receiving encrypted content from servers (or alternatively the parties are exchanging encrypted communication between themselves). All parties share a group key which is managed by a group controller (GC). We assume that the GC can communicate with each of the

[1]If the usage rules are of the type "User A can get the content and do anything he wants with it" then encryption can be the only governance mechanism that is used. If the rules are more complex. For instance, "User B can use the content at most three times" then more complex mechanisms should be used (e.g. based on tamper resistance), but typically encryption is used as a first line of defense.

other parties using secure one-to-one channels, which are realized using standard encryption and authentication techniques. In order to do so the GC typically shares a different key with each of the users.

The content servers deliver the content encrypted with the group key, to ensure that only group members can use it. The system therefore enforces the rule "Group members are allowed to use content sent by content servers", since knowledge of the group key enables decryption of the content. This system can model a content subscription group, namely where members of the group U are subscribers which are allowed to use the content.

In order to enforce the usage rules the group key must be changed when users join or leave the group. This is essential in order to

- Prevent leaving members from decrypting content that will be sent to the group in the future.
- Prevent joining members from decrypting content that was previously sent to the group (namely, provide *backward secrecy*).

Joins are usually trivial to handle. When a user u' wants to join U the GC should pick a new group key, send a message to the group containing the new key encrypted with the old group key, and also inform u' of the value of the new key. (There are many ways in which these messages can be sent, but they are mostly irrelevant for the discussion of this paper.) All further content sent to the group should be encrypted with the new key, until other members join or leave the group. This procedure supports backward secrecy and prevents u' from decrypting old content that was sent to the group. (We note that in the LKH key management scheme, which is the focus of this paper, the join operation is more complicated. We discuss this issue below.)

The main design challenge is to efficiently handle events in which users leave the group, or are forced by the GC to leave the group (say, because they violated usage rules). When a user u leaves the group a new group key should be generated by the GC and become known to every user remaining in the group. The keys known to u should be revoked in a way which prevents u from learning any information about messages encrypted with the new group key.

There is of course a trivial method for rekeying the group in the case of a leave: The GC chooses a new group key k, and sends it independently and privately to each of the users, except for the leaving member. For example, the GC can share a different key with each user, and encrypt k using this key. The problem with this approach is that the GC has to send a total of $n-1$ encryptions (the length of each is of the same order as the length of the new key). The total length of the messages it has to send is therefore $O(n)$ keys (or $O(n|k|)$ bits) and might be too large if the number of group members n is large (say, a few millions).

1.1 The LKH Scheme

A very appealing user revocation method was suggested in [16, 17]. In this method, commonly denoted as LKH (Logical Key Hierarchy), the GC associates a binary tree with the group, and associates each user with a different *leaf* of this tree. Therefore for a group of n users the tree is of depth $d = \log(n)$. The tree is used for key management and is not

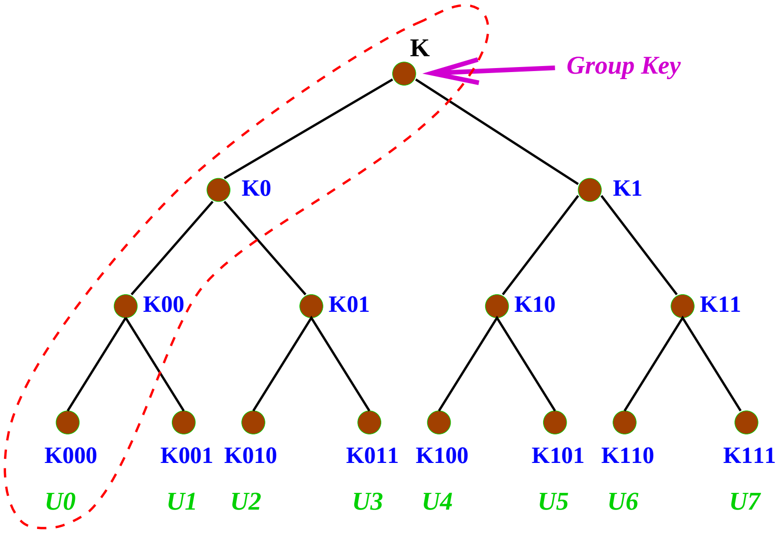

Figure 1: The LKH key data structure (the keys of U_0 are encircled).

used as part of the mechanism in which messages are sent to users (in particular, it has no relation with distribution trees used in multicast communication).

The GC associates a random key with each node of the tree, and therefore knows the keys of all the nodes in the tree. The GC also provides each user with all the keys in the path from the user's leaf to the root. Since all these paths converge at the root of the tree, every user knows the root key and therefore this key can serve as the group key. In the example depicted in Figure 1, user U_0 is associated with the leftmost leaf and knows keys K, K_0, K_{00} and K_{000}. The root key K is the group key.

When a user is removed from the group the GC must change all the keys in the path from this user's leaf to the root. All the users that remain in the group must update their keys, namely change the keys in the intersection between the path from their leaf to the root and the path from the removed user's leaf and the root. In particular, this means that every remaining user learns the new root key. The new root key is then used as the new group key. The update of the users' keys can be done by the GC sending a single message that contains an encryption of $2\log(n) - 1$ keys. (The details of this message appear in Appendix A and are not important for understanding the rest of this manuscript.) Improved schemes in [3, 11] reduce this overhead to a single message with $\log(n)$ encryptions of keys, i.e. with total length of $\log(n)|k|$ bits.

If backward secrecy is required then a join operation in the LKH scheme is similar to a remove operation in the sense that the keys that the joining user receives must be different than the keys previously used in the system. This is required in order to prevent the joining user from learning previous messages that were sent to group encrypted with the group key (it is not sufficient to change only the root key because other keys in the path might have been used in previous revocation messages to encrypt the root key itself or information that reveals it). The joining user is assigned a leaf, and all the keys in the path from this leaf to the root must be updated, using the same method and the same overhead as in user deletion.

1.2 The State Update Problem

The LKH method is quite efficient: each user has to keep a personal key with $\log n$ keys, i.e. of length $\log(n)|k|$ bits, and the length of a key update message is also $\log(n)|k|$ bits. The main drawback of the basic LKH method and its variants is the requirement that group users update their state whenever users join or leave the group. Suppose that t users join or leave the group and that a user was offline when these updates took place. This user must now update the keys in the path from its leaf to the root in order to be able to process future update messages. If the user also needs to decrypt content that was sent during the period it was offline, it must also learn the t group keys that were in effect during that period. The straightforward way for the user to learn this information is for it to receive and process all t key update messages. The total length of these messages is $t\log(n)|k|$ bits. Typical values for these parameters are, say, $n = 1,000,000$, $t = 1000$, and $|k| = 128$. In this case the total overhead is about $2,500,000$ bits. The computation overhead is almost always negligible, since it only involves efficient "private key" cryptographic operations.

The LKH method was suggested in the context of secure multicast, that builds secure communication on top on the Internet's multicast layer. In that context the state update problem must be addressed since multicast communication is lossy and therefore the key update messages might not be received by all users.

The state update problem might be even more relevant in a digital rights management setting. Users might be offline most of the time. Content can be delivered to them separately from the rules and cryptographic keys that enable its use. The content itself can be delivered in different channels such as a webcast, web servers, a peer-to-peer network, or in static devices such as CDs, DVDs, or other similar types of media. The rules are typically delivered to users when they connect directly with servers, such as the GC. In this setting it is natural that users do not have continuous communication with the GC, but rather contact it from time to time. Users might acquire content while being disconnected from the GC (e.g. by users "beaming" music from one device to the other), and when they connect with the GC they should obtain keys that enable access to that content[2]. Since key updates might occur frequently, it is reasonable to assume that during the period in which a user is offline there are several updates to the group key. Once the user is online again it should get the group keys that were used since the last time it was connected.

We reduce the overhead of users who were offline and need to update their state. Namely, the overhead of learning the group keys that were sent in key update messages which were not received, and also updating the personal keys.

This task can be accomplished by a "universal" solution that applies to all users. For example by making all the key update messages of the LKH scheme available on a web site or constantly retransmitted, or by using one of the stateless methods of [10] and attaching a header to every message (or posting the header on a web site).

We are able to make the update data much shorter by using a different update message per user, depending on the period during which the user was offline and the content which it is allowed to receive. In particular, the personal key of every user contains only $\log n$ keys as in the LKH scheme, and the update message is of length $O(\log t)$ keys.

[2] "Beamed" content might have a rule that enables it to be used a few times for free, but require that a user should get a key from the GC in order to use the content more times.

1.3 Contributions

We address the problem of making state updates as efficient as possible, mostly from the perspective of the communication between the GC and the user. Consider a group U in which there are n users, that uses the LKH scheme for key management. Consider now a user u who was offline during a period in which t users were removed from the group. The length of a key is $|k|$. The trivial state update message contains all the key update messages that were sent during the t key updates, and its length is $t\log(n)$ keys or $t\log(n)|k|$ bits. A naive analysis of the run time reveals also that given this message the user should perform $O(t\log(n))$ decryptions in order to update its state.

The first trivial improvement is to note that after being offline there is no need for the user to learn the actual key update messages. The information that it should learn consists of the group keys that were used while it was offline, which are needed in order to decrypt messages that were sent when these keys were in effect, and the current keys of the nodes in the path from the user's leaf to the root, which are required in order to process future key update messages. The total is t group keys, and $\log(n)$ node keys, which can be sent in a message of length $t + \log(n)$ keys (this message is directed to a specific user, rather than being a universal message that can be used by all users).

In the rest of the paper we present additional improvements:

- Group keys can be generated in a method that enables a concise representation of a sequence of consecutive keys. This enables a list of t consecutive keys to be sent using a shorter message. Namely,
  - t consecutive keys can be sent using a message of length $O(1)$ keys ($2|k|$ bits). This method is insecure against collusions between users.
  - t consecutive keys can be sent using a message of length $O(\log t)$ keys ($2\log t|k|$ bits). This method is secure against collusions between any number of users.
- We observe that not all the keys in the path from the user's leaf to the root are changed by the update messages, but rather only keys that intersect with the paths from the leaves of deleted or joining users. A probabilistic analysis shows that with high probability only $O(\log t)$ of the $\log n$ keys in the path should be updated. Similarly, the expected number of keys that have to be changed is $\log(t) + \log\log(n/t)$.

To sum up, the communication overhead of updating the state of a user which was offline during t key updates is reduced, with high probability, from $O(t\log(n)|k|)$ bits to $O(\log t|k|)$ bits. If we do not care about security against collusions of users then this overhead is composed of $O(\log t|k|)$ bits for updating the path, and $O(|k|)$ for sending old group keys. If security against user collusions is required, the overhead is $O(\log t|k|)$ bits for each of these tasks.

1.4 Related Work

User revocation schemes can be traced back to the *broadcast encryption* scheme of Fiat and Naor [5]. This system enables the removal of any number of users as long as a limited number of them collude (the number of colluding users must be smaller than a system parameter which affects the overhead).

The Logical Key Hierarchy scheme (LKH), which was suggested independently by Wallner et. al [16] and Wong and Lam [17], enables to revoke any number of users with security against any number of colluding users. The motivation for this scheme was providing security for multicast groups. Since the scheme requires users to change their keys (state) whenever other users leave or join the group, the users must be connected most of the time. The communication overhead of the LKH scheme was improved in [3, 11] by a factor of 2 (see discussion in Appendix A), and a better join mechanism was suggested in [15].

In the information theoretic scenario, Luby and Staddon [9] provide lower bounds for any revocation algorithm. Kumar et. al [7] design a *one-time* revocation system for removing s users in which the message length is $O(s \log n)$ and the length of the personal key is $O(s^2)$ and does not depend on n. Since this system is good for a single revocation the question of updating the user's state does not exist. Other, more efficient systems for one-time revocation, based on polynomial interpolation, were by suggested Anzai et. al [1] and Naor and Pinkas [12]. In these schemes the length of the revocation message is s keys and the personal key contains only a single key. The scheme of [12] can be generalized for many revocations, and provides traitor tracing capabilities.

The MARKS system [2] is a key assignment method that addresses multicast scenarios in which premature removal of users is rare and it is known in advance what content each user is allowed to obtain. It is assumed that in these scenarios there is no need for revocation messages. It is further assumed that users subscribe in advance for a sequence of consecutive "content" events, e.g. for a pay-TV movie (which is composed of consecutive minutes). Fine granularity is achieved by dividing time into short "application data units", ADUs, (e.g. an ADU being a minute of a video) and providing a different key for every ADU. A user should receive the keys of the ADUs which it is entitled to use. The key assignment to ADUs is done using the same method we suggest in Section 2.2 (however, no proof of security or rigorous complexity analysis is given in [2]). A more general tree based construction is presented at [18].

A recent approach taken by Naor, Naor and Lotspiech [10] is to design a system in which users can be completely *stateless*. That is to say that users do not have to change their personal data when revocations or joins take place. Instead of requiring users to change their state, the GC attaches to each message a header which depends on the list of active users, and enables only these users to decrypt the message (the header information is similar to the list of key update messages that must be available to all users at all times if the LKH scheme is used). They suggest two new key update algorithms, which require each user to store a personal key of $\log n$ and $\frac{1}{2}\log^2 n$ keys, respectively. After t key updates the length of the header information is $t \log n$ and t keys, respectively. The second scheme is very efficient in terms of the length of the header information, which does not even depend on n, at the cost of increasing the length of the personal key. (Note however that the overhead analysis of this scheme requires n to be an upper bound on the number of users *throughout* the lifetime of the system.)

2 A Concise Representation of Keys

Let us denote the group key that is used between the key updates i and $(i+1)$ as k_i. A user which did not receive the t key update messages numbered $i, i+1, \ldots, j$, where $j-i = t-1$, must learn keys $k_i, \ldots, k_j$ in order to decrypt content that was sent during these periods. This can be trivially achieved using a message containing these t keys. We show below two methods of reducing this overhead to two keys and $O(\log t)$ keys, respectively.

The methods described in this section can be used in more general scenarios. They enable to generate n pseudo-random keys in a way that enables short representations of any subset of *successive* keys, while preserving the pseudo-randomness of the other keys.

2.1 A Method with No Security Against Collusions

Let N be a predefined constant (say, $N = 10,000$), and let F be a pseudo-random generator with input length of $|k|$ bits and output length of $2|k|$ bits. The system uses a seed of length $2|k|$ bits that can be used to send update messages for N key updates. Afterwards new seeds should be generated.

The system operates in the following way. The GC chooses in advance two seeds, L_1 and R_N, each of length k. Denote by $F_0(x)$ and $F_1(x)$ the left and right halves of the output of F. The GC defines the following values

$$
\begin{aligned}
L_i &= F_0(L_{i-1}) \ \ i = 2, \ldots, N & (1)\\
R_i &= F_0(R_{i+1}) \ \ i = N-1, \ldots, 1 & (2)\\
k_i &= F_1(L_i) \oplus F_1(R_i) \ \ i = 1, \ldots, N & (3)
\end{aligned}
$$

The key k_i is used as the group key after the ith key update[3]. Note that given L_i and R_j, with $i < j$, one can compute all the keys $k_i, \ldots, k_j$. The update message to a user that did not receive key update messages i through j consists therefore of L_i and R_j alone, is of length $2|k|$ bits, and enables the user to compute $k_i, \ldots, k_j$.

The following lemma states that given a pair of keys $\langle L_i, R_j \rangle$ (and no other L or R values), the set of keys $\{k_\ell \,|\, \ell \notin [i,j]\}$ which are generated by this scheme is pseudo-random. The lemma is used to prove a theorem stating that a user that receives $\langle L_i, R_j \rangle$, and might know the values of some other keys k_ℓ with $\ell \notin [i,j]$, does not learn anything about other keys which are not in $[i,j]$.

Lemma 1 *Given only L_i and R_j, with $i < j$, the set of keys k_ℓ, $\ell \notin [i,j]$ is pseudo-random.*

Proof: The proof uses a standard hybrid argument. Suppose that it is possible to distinguish between the keys $\{k_\ell \mid \ell \notin [i,j]\}$ and a random sequence. Namely there is a distinguisher (i.e. a probabilistic Turing machine) for which there is non-negligible difference (denoted by δ) between the probability that the distinguisher outputs 1 in each of these cases. We can then construct a distinguisher between the output of F and random values. Our distinguisher is given a pair (x_0, x_1) which is either $F(y)$ for a randomly chosen y, or a random $2|k|$ bit string.

[3] In the improvement of LKH described in [3, 11] the root key is defined by a different method and cannot be set to an arbitrary value. Therefore, the root key should be used to encrypt k_i, which should be used as the group key.

Suppose first that $j = N$. We construct $N - t$ hybrids. Hybrid H_ℓ (for $1 \leq \ell \leq i$) is defined in the following way:

- Set $k_1, \ldots, k_{\ell-1}$ to random values.
- Set L_ℓ to a random value.
- Set R_N to a random value, and define all other keys using L_ℓ and R_N.

The distribution of keys $(k_1, \ldots, k_{i-1})$ in H_1 is identical to that generated by the construction, whereas the distribution of these keys in H_i is random. The difference therefore between the probability that the user outputs 1 given H_1 and given H_i is δ. This means that there is an $1 \leq \ell < i$ for which the difference between the probabilities associated with H_ℓ and $H_{\ell+1}$ is at least $\delta/(N-t)$ and is non-negligible. Our distinguisher algorithm for F picks a random location $1 \leq \ell' < i$, and performs the following operations:

- Sets $k_1, \ldots, k_{\ell'-1}$ to random values.
- Sets $L_{\ell'+1}$ to x_0, and defines L_s for $s > \ell' + 1$.
- Sets R_N to a random value, and defines R_s for $\ell' \leq s < N$.
- Sets $k_{\ell'}$ to $x_1 \oplus R_{\ell'}$.
- For $\ell' + 1 \leq s \leq N$ defines $k_s = L_s \oplus R_s$.

With probability at least $1/(N-t)$, $\ell' = \ell$ for which the difference in probabilities is $\delta/(N-t)$. Therefore our distinguisher succeeds with probability at least $\delta/(N-t)^2$.

Assume now that $j < N$ and $i > 1$. The argument used above can be used with hybrids ranging over all the locations $\ell \notin [i,j]$ and yields the same result. $\Box$.

Theorem 1 *Given L_i and R_j, with $i < j$, any set P of indices ($P \subseteq [1, i-1] \cup [j+1, N]$), and the keys $\{k_s \mid s \in P\}$, the set of keys $\{k_\ell \mid \ell \notin P \cup [i,j]\}$ is pseudo-random.*

Proof: We show that if the theorem does not hold then neither does lemma 1.

Suppose that the theorem does not hold. Then there is a distinguisher D that receives as input the following values:

- Indexes i, j and values L_i, R_j.
- A set of keys $\{k_\ell \mid \ell \notin [i,j]\}$
- A set of indexes P, and an assurance that the keys $\{k_\ell \mid \ell \in P\}$ were generated according to the construction.

Define T as the set of indexes that are not in P or in $[i,j]$. The algorithm is able to distinguish between the event that the keys indexed by T are random, and the case that they were generated according to the construction.

Assume that we are given an input instance to the distinguishing problem defined in lemma 1. Namely, indexes i, j and values L_i, R_j, and a set of keys $T = \{k_\ell \mid \ell \in [1, i-1] \cup [j+1, N]\}$. We should distinguish between the case that the keys in T were generated according to the construction (event C), and the case that they are random (event R).

Define two experiments:

- **T1**: Pick a random set P of indexes not in $[i,j]$. Feed the input and P to the distinguisher D.
- **T2**: Pick a random set P of indexes not in $[i,j]$. Replace the keys $\{k_\ell \mid \ell \notin [i,j] \cup P\}$ with random values. Feed the input and P to the distinguisher D.

Define $Q_{C,T1}$ as the probability that D outputs 1 in experiment T1 given event C. Define $Q_{R,T1}$ as the probability that D outputs 1 in experiment T1 given event R. Similarly, define $Q_{C,T2}$ and $Q_{R,T2}$.

Since we assume the theorem not to hold, $Q_{C,T1}$ is far from $Q_{C,T2}$. It is also obvious that $Q_{R,T1} = Q_{R,T2}$, since in both cases the keys not in $[i,j]$ are random.

It therefore cannot be the case that both of the following two events happen: $Q_{C,T1}$ is close to $Q_{R,T1}$, and $Q_{C,T2}$ is close to $Q_{R,T2}$. Assume, without loss of generality, that $Q_{C,T1}$ is far from $Q_{R,T1}$. Then $T1$ is a distinguisher that contradicts lemma 1. $\square$

The method suggested here is not immune to collusions between two corrupt users receiving update messages. For example consider user A which paid for content during times $[1,100]$ and user B which paid for content during times $[201,300]$. Suppose now that user A was offline during times $[50,70]$, and user B was offline during times $[250,270]$. User A contacts the GC and receives L_{50} and R_{70}, and user B contacts the GC and receives L_{250} and R_{270}. Now the two users can use L_{50} and R_{270} together to compute the keys $k_{50},\ldots,k_{270}$. In particular, they can compute the keys $k_{101},\ldots,k_{200}$ which neither of them is entitled to receive. The same attack can also be run by a single user that receives two update messages (e.g. consider the above example with A and B being the same user who did not pay for receiving the content during times $[101,200]$). For this reason the scheme must not send two or more updates to a single user, if between two periods in which the user was offline there is a period in which it is not entitled to obtain the group keys.

The communication overhead of the method consists of an update message that contains two keys, namely of length $2|k|$ bits, as long as the sequence of keys that the user should learn is within a single "block" of keys (i.e. generated from the same seeds). If the sequence of keys contains keys from c blocks, then the communication overhead is $2c|k|$ bits. (However, if n is sufficiently long then c is typically very small, i.e. $c = 1$ or 2.)

2.2 A Method Secure Against Collusions

The following key generation method supports short update messages which are secure against collusions of any set of corrupt users. Let N be a predefined constant which is a power of 2 (say, $N = 2^{20}$). The method enables to generate N group keys (of length $|k|$ bits each) while enabling to compute any t *consecutive* keys from at most $O(\log t)$ values of length $|k|$ each. If the GC generates the group using this method then a user which does not receive t key update messages can receive a message of length $O(\log t)$ keys from the GC and use it to reconstruct the t group keys that were sent in the key update messages.

The keys are generated in the following way (similar to the Goldreich-Goldwasser-Micali [6] or the Naor-Reingold [13] constructions of pseudo-random functions).

- Let F be a pseudo-random generator with input of length $|k|$ bits and output of length $2|k|$, and denote by $F_0(x), F_1(x)$ the left and right halves of the output of F for an input x.

- Imagine a full binary tree of depth $\log(N)$ which has N leaves. The GC chooses a random key of length $|k|$ for the root node, and defines a key for every other node of the tree in the following way, going from the root down: Let v be a node, and let $v0, v1$ be its two sons. Denote by k_v the key of node v. Then $k_{v0} = F_0(k_v)$ and $k_{v1} = F_1(k_v)$.
- The key of the ith leaf is used as the group key after the ith key update.

The construction ensures that a key of a node v enables to compute the keys of all the leaves of the subtree rooted in v, using the same key computation method that was used in the initial generation of the tree. (Generating the keys in the leaves of the subtree requires keeping at most $\log n$ internal key values in memory, and doing an amortized computation of $O(1)$ applications of F per key.) The following theorem states that this key generation method is secure.

Theorem 2 *Given any set S of leaves, and the values of the keys of a set of nodes R (either internal nodes or leaves) such that S is exactly the union of the leaves of the subtrees rooted by nodes in R, the values of the other nodes of the tree are pseudo-random.*

Proof: The proof is based on the proof of the pseudo-randomness of the GGM construction in [8]. Assume that there is a distinguisher that distinguishes between the values of the leaves of the tree and random values, and construct an adversary that distinguishes between the output of the pseudo-random generator F and random values, based on queries that it makes to values of leaves of the tree.

The adversary is given a pair of values (x, y) and it should distinguish between the case that they are the output of F, and the case that they are random. The adversary constructs a full binary tree, chooses a random value to its root and sets the values of the nodes according to the construction described above. Define S' as the set of nodes in R, their ancestors and their descendants (namely nodes in the paths from R to the root, and nodes in subtrees rooted by nodes in R). The values of the nodes in S' are then fixed and will not be changed. The set S of leaves is exactly the set of leaves in S'.

The values of the other nodes are defined during the interaction with the distinguisher, based on a hybrid construction, as in the proof in [8]. During the interaction the distinguisher asks for the values of different leaves. In order to compute a value of a leaf, the values of the nodes in the path from the root to the leaf must be computed, if they were not computed before.

A node is defined as "marked" if it is in S', or its value was computed in a previous step of the interaction. The ith hybrid is defined by (1) providing the values for the first $i-1$ unmarked nodes according to the construction, (2) defining the values of the descendants of the ith unmarked node to be (x, y), (3) providing random values for the other marked nodes. (Note that the order of the nodes is defined by the queries to leaf values that are made in the interaction. This order might be different in different interactions. Furthermore, the value of a leaf that is provided in the interaction might affect the order of the nodes which are unmarked at that stage.)

The number of hybrids is bounded by N. The last hybrid is the tree defined by the construction, and therefore the probability that the distinguisher outputs 1 given this hybrid is equal to this probability in the case the tree is built according to the construction. In the first hybrid all leaves except those in S have random values, and therefore the probability that the distinguisher outputs 1 given this hybrid is the same as if the tree (except the leaves in S) is random. Therefore if the difference between these two probabilities is δ, there is a hybrid which distinguishes between the output of the generator and a random value with probability at least $\delta/(2N)$. $\square$

The key update method: Consider a user which did not receive the messages of t successive key updates and needs to learn the keys of the t successive leaves which were associated with the group keys sent in these key updates. Denote this set of leaves as S. In order to enable the user to learn these keys it is sufficient to provide it with the keys of a minimal set R of nodes, such that S is *exactly* the union of the leaves of the subtrees rooted by the nodes in R. The GC should therefore send to the user the keys of the nodes in R, and the user can use these keys to compute the group keys.

Theorem 3 proves, using an inductive argument, that for every sequence S of t *successive* leaves there is such a set R of at most $O(\log t)$ nodes. The length of the message sent to the user is therefore $O(\log(t)|k|)$ bits.

In order to analyze the size of R we first prove the following lemma. (The same lemma is also widely used in computational geometry, see e.g. [4, 14].)

Lemma 2 *Let T be a complete binary tree with $N = 2^n$ leaves. Then given any set S of* consecutive *leaves, there is a set R of at most $2n-2$ nodes such that S is exactly the union of the leaves of the subtrees rooted by the nodes in R.*

Proof: The proof is by induction. Let T_i be a complete binary tree with 2^i leaves. Define the following two values for T_i:

- R_i is the maximum, taken over all sets S of consecutive leaves, of the size of the minimal set of nodes R such that S is exactly the union of the leaves of the subtrees rooted by the nodes in R.

- E_i is defined in a similar way, but taking the maximum over all sets S that contain either the leftmost or the rightmost leaf of T_i. Namely, E_i is the maximum, taken over all sets S of consecutive leaves that contain the leftmost or the rightmost leaf of T_i, of the size of the set of nodes R such that S is exactly the union of the leaves of the subtrees rooted by the nodes in R.

It holds that

$$R_i = \max(R_{i-1}, 2E_{i-1}) \tag{4}$$
$$E_i = \max(E_{i-1}, E_{i-1}+1) = E_{i-1}+1 \tag{5}$$

Equation 4 holds since there are only two options for the sequence S that maximizes R_i: If it is contained in one half of the tree (e.g. in the left half rooted by the left son of the root of T_i) it is actually a sequence of leaves in a tree of depth $i-1$ and the number of nodes in R is bounded by R_{i-1}. If S is contained in both halves of the tree, it is a union of two sequences in two trees of depth $i-1$, where each of these sequences contains either the leftmost or the rightmost leaf of its subtree. The size of R is then bounded by $2E_{i-1}$.

Equation 5 holds since the sequence S that maximizes E_i is either contained in one half the tree (in which case the size of R is bounded by E_{i-1}), or contains one half of the tree and in addition a sequence of leaves from the other half. In this case R contains the root of the first half, and at most E_{i-1} additional nodes.

For a tree of depth 1 (namely with two leaves), $R_1 = E_1 = 1$. It therefore holds that

$$R_i = 2E_{i-1} = 2i - 2 \tag{6}$$

$$E_i = i \tag{7}$$

This proves the lemma. (Note that if instead of using a binary tree the tree has more descendants per node, the overhead increases.) □

The lemma provides the same bound for any sequence S regardless of the number of leaves it contains. The following theorem provides a bound which depends on $|S|$.

Theorem 3 *Given any set S of t consecutive leaves in a complete binary tree, there is a set R of at most $2\lfloor\log(t)\rfloor + 1$ nodes such that S is exactly the union of the leaves of the subtrees rooted by the nodes in R.*

Proof: Let $N = 2^n$ be the number of leaves in the tree. Let $r = \lfloor\log(t)\rfloor$, namely the largest power of 2 which is not greater than t. We consider the tree as a collection of $N/2^r$ subtrees of depth r, with 2^r leaves in each subtree.

Since it holds that $t/2 < 2^r \leq t$ the sequence of leaves in S can span at most three consecutive such subtrees. The leaves of the inner subtree are completely contained in S, whereas some or all of the leaves of the outer subtrees are contained in S. Applying lemma 2, the number of nodes that cover the leaves in S is at most $E_r+1+E_r = 2r+1 = 2\lfloor\log(t)\rfloor+1$.
□

3 Updating Keys on the Path from a Leaf to the Root

The fact that a user is offline during t key updates does not necessarily mean that all the keys in the path from the user's leaf to the root were changed by the key updates. In fact, if the locations of the users that are leaving or joining are random, then for each key update there is a 1/2 chance that only the root key is affected, a 1/4 chance that two keys on the path to the root are affected, and in general a $1/2^i$ chance that i keys on that path are changed. (The assumption of random revocation locations is justified if the system associates users with leaves at random, and we assume that users are revoked independently of the location to which they were mapped in the tree.)

The update message from the GC to the user should contain only the keys that need to be updated, rather than all the $\log n$ keys in the path from the user's leaf to the root. In this section we prove the following theorem:

Theorem 4 *For any user, after t key updates of random leaf keys using the LKH protocol,*

- *It holds with high probability that $\log t + O(1)$ keys need to be updated, and*
- *The expected number of keys that have to be updated is $\log(t) + \log\log(n/t) + O(1)$.*

Proof: Fix a certain leaf (the leaf of the user who was offline), and assume that the key updates are applied to randomly chosen users. The probability that the intersection between the path from the user to the root, and the path from an updated key to the root, contains exactly i nodes is 2^{-i}. This is also the probability that the length of the intersection is strictly greater than i.

Considering t key updates, the probability that the intersection with all the paths from the leaves of the updated keys are of length at most ℓ, is $(1-2^{-\ell})^t$. Setting $\ell = \log t + c$, yields that the probability of an intersection of length at most ℓ is $(1-2^{-\ell})^t = (1-\frac{1}{t2^c})^t \approx e^{-2^{-c}} \approx 1-2^{-c}$. In particular, $\ell = \log t + 2$ yields a probability of $(1-\frac{1}{4t})^t = 0.78$. Similarly setting $\ell = \log t + 3$ and $\ell = \log t + 4$ yield probabilities of 0.88 and 0.94, respectively. The length of the intersection is therefore greater than $\log t + 4$ nodes with probability at most 6%.

As for the calculation of the expected length of the intersection, let us again set $\ell = \log t + c$, where c is a parameter whose value we will define below. The probability that all the t intersections are of length ℓ or less is $(1-2^{-\ell})^t = (1-\frac{1}{t2^c})^t = e^{-2^{-c}}$. The expected length of the intersection with all paths is therefore bounded by

$$(\log t + c) \cdot e^{-2^{-c}} + \log n \cdot (1 - e^{-2^{-c}}).$$

Assuming that 2^c is large, we can use the following approximation

$$\begin{aligned} &\approx (\log t + c) \cdot (1 - 2^{-c}) + \log n \cdot 2^{-c} \\ &= \log t + c + 2^{-c} \log(n/t) - 2^{-c} c \end{aligned}$$

Setting $c = \log\log(n/t)$ cancels out the third element and reveals that the expectation is $\log t + \log\log(n/t) + O(1)$. This is therefore the expected length of the update message. □

Acknowledgments

We would like to thank Michael Atallah and Bernard Chazelle for referring us to the computational geometry origins of lemma 2.

References

[1] J. Anzai, N. Matsuzaki and T. Matsumoto, "A Quick Group Key Distribution Scheme with Entity Revocation", *Adv. in Cryptology – Asiacrypt'99*, Springer-Verlag LNCS 1716 1999, pp. 333–347.

[2] B. Briscoe, "MARKS: Zero side effect multicast key management using arbitrarily revealed key sequences", *Proc. First International Workshop on Networked Group Communication (NGC'99)*, 1999.

[3] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor and B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions", *In Proc. INFOCOM '99*, Vol. 2, pp. 708-716, New York, NY, March 1999.

[4] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, Germany, 1987.

[5] A. Fiat and M. Naor, "Broadcast Encryption", *Adv. in Cryptology – Crypto '93*, Springer-Verlag LNCS vol. 773, 1994, pp. 480–491, 1994.

[6] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions", *J. of the ACM*, Vol. 33, No. 4, 1986, pp. 792-807.

[7] R. Kumar, S. Rajagopalan and A. Sahai, "Coding constructions for blacklisting problems without computational assumptions", *Adv. in Cryptology – Crypto '99*, Springr-Verlag LNCS 1666, pp. 609–623, 1999.

[8] M. Luby, *Pseudorandomness and Cryptographic Applications*, Princeton Computer Science Notes, 1996.

[9] M. Luby and J. Staddon, "Combinatorial Bounds for Broadcast Encryption", *Adv. in Cryptology – Eurocrypt '98*, Springer-Verlag LNCS 1403, 1998, pp. 512–526.

[10] D. Naor, M. Naor and J. Lotspiech, "Revocation and tracing schemes for stateless receivers", *Adv. in Cryptology – Crypto '01*, Springer-Verlag LNCS 2139, 2001, pp. 41–62.

[11] D. McGrew, A. T. Sherman, "Key establishment in large groups using one-way function trees", *IEEE Transactions on Software Engineering*, vol. 29, no. 5 (May 2003), 444-458.

[12] M. Naor and B. Pinkas, "Efficient Trace and Revoke Schemes", *Proceedings of Financial Crypto '2000*, February 2000.

[13] M. Naor and O. Reingold, "Number-Theoretic constructions of efficient pseudo-random functions", *Proc. 38th IEEE Symp. on Foundations of Computer Science*, 1997, pp. 458–467.

[14] F.P. Preparata and M.I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, New York, 1985.

[15] M. Waldvogel. G. Caronni, D. Sun, N. Weiler and B. Plattner, "The VersaKey Framework: Versatile Group Key Management", *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 9, Sep. 1999, pp. 1614–1631.

[16] D. M. Wallner, E. G. Harder and R. C. Agee, "Key Management for Multicast: Issues and Architecture", RFC 2627, June 1999.

[17] C. K. Wong, M. Gouda, and S. S. Lam, "Secure Group Communications Using Key Graphs", *Proc. of SIGCOMM '98*, pp. 68–79.

[18] A. Wool, "Key Management for Encrypted Broadcast", *ACM Trans. on Information and System Security*, Col. 3, No. 2, 2000. pp. 107–134.

A The LKH Scheme

Tree based group rekeying schemes were suggested by Wallner et. al [16] (who used binary trees), and independently by Wong et. al [17] (who considered the degree of the nodes of the tree as a parameter). We concentrate on the scheme of [16] since binary tress require a smaller communication overhead per user revocation. When this scheme is applied to a group of n users it requires each user to store $\log n+1$ keys. It uses a message with $2\log n-1$ key encryptions in order to delete a user and generate a new group key. This process should be repeated for every deleted user. The scheme has better performance than the Fiat-Naor [5] scheme when the number of deletions is not too big. It is also secure against any number of corrupt users (they can all be deleted from the group, no matter how many they are). A drawback of the scheme is that if a user misses some control packets relative to a user deletion operation (e.g., if it temporarily gets disconnected from the network), it needs to ask for the missed control packets. This also applies for a user who misses join operations if the scheme is set to support backward secrecy.

We now describe the scheme of [16]. Let $u_0, \ldots, u_{n-1}$ be n members of the group (in order to simplify the exposition we assume that n is a power of 2). They all share a group key k with which group communication is encrypted. There is a single group controller, which might wish at some stage to delete a user from the group and enable the other members to communicate using a new key k', unknown to the deleted user.

The group is initialized as follows. Users are associated with the leaves of a tree of height $\log n$ (see Figure 1). The group controller (GC) associates a key k_v with every node of the tree, and sends to each user (through a secure channel) the keys associated with the nodes along the path connecting the user to the root. For example, in the tree of Figure 1, user u_0 receives keys k_{000}, k_{00}, k_0 and k. Notice that the root key k is known to all users and can be used as the group key and encrypt group communication.

In order to remove a user u from the group the GC performs the following operations. For all nodes v along the path from u to the root, a new key k'_v is generated. New keys are encrypted as follows. Key $k'_{p(u)}$ is encrypted with key $k_{s(u)}$, where $p(u)$ and $s(u)$ denote respectively the parent and sibling of u. For any other node v along the path from u to the root (excluded), key $k'_{p(v)}$ is encrypted with keys k'_v and $k_{s(v)}$. All encryptions are sent to the users. For example, in order to remove user u_0 from the tree of Figure 1 the following set of encryptions is transmitted (see Figure 2): $E_{k_{001}}(k'_{00}), E_{k'_{00}}(k'_0), E_{k_{01}}(k'_0), E_{k'_0}(k'), E_{k_1}(k')$. It is easy to verify that each user can decrypt only the keys it is entitled to receive. If backward secrecy is required then a user join operation is similar to user removal (see Section 1.1). The update of the keys in the path from the leaf of the joining user to the root is performed in a similar manner to the key update in the case of user removal.

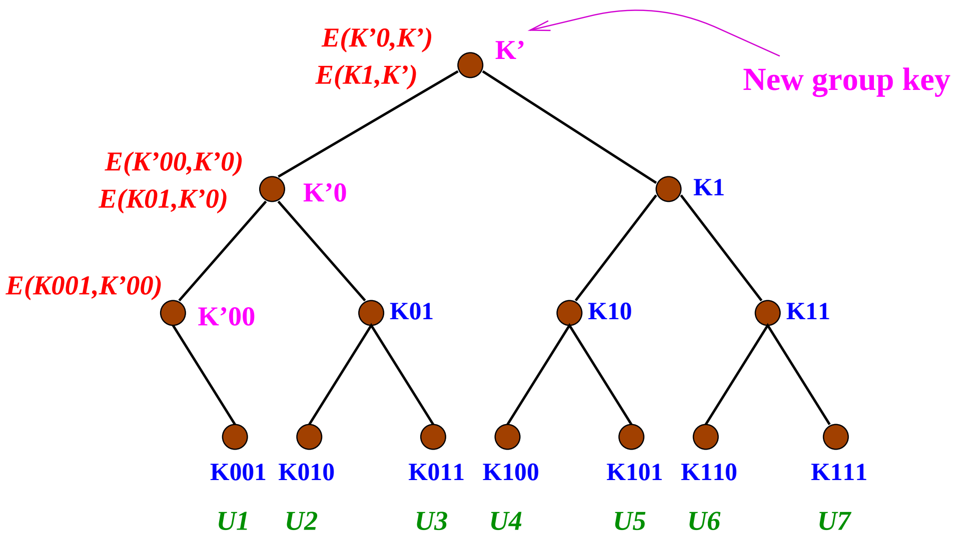

Figure 2: The delivery of new values to the keys surrendered to u_0.