Tree-Based Group Diffie-Hellman (TGDH) Protocol

Idea behind Group Keys

In an organization, there may be several groups of people that should be privy to certain information, and these groups could be bigger than 2 people. Up until now, we have discussed two individuals communicating with either a shared secret key that no one else knows, but never entertained the idea of several people (more than 2) having a single shared secret key. Furthermore, organizations may have several groups with different levels of access/security. Thus, instead of just having one secret key shared with one other individuals, it makes sense for there to be several secret keys, one for each group an individual belongs to.

With this added functionality comes added flexibility:

1) How do we add someone to an existing group?

2) What happens if someone has to leave a group? (Namely: we wouldn't want that person to continue to be able to read the group communications.)

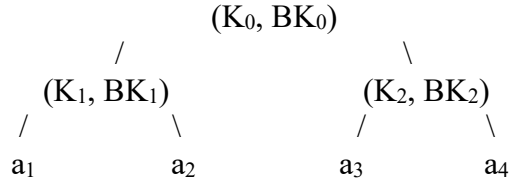
3) As these changes may update quite a few people, is there an efficient way so that we don't have to update too many keys whenever some change occurs to a group?

General Idea Behind a Group Diffie-Hellman Key

If there are three users, Alisha, Bennie and Calista with private keys a, b, and c, then g^{abc} mod p can be their shared key. (Notice that it would still be okay in this situation if the shared key between Alisha and Bennie was g^{ab} mod p, the shared key between Alisha and Calista was g^{ac} mod p, and the shared key between Bennie and Calista was was g^{bc} mod p.

Use of a Binary Tree to implement Tree-Based Group Diffie-Hellman

Imagine that the structure of the groups could be organized in a binary tree. All end users would be leaf nodes and the shared secret keys between groups would be stored in internal nodes. This idea is best understood with a diagram. The whole system uses a single public key prime, p and generator, g. Each user (leaf node) has its own secret key, a_i. Each internal node, j, has two different keys associated with it: K_j, which represents the shared secret key between all users in the subtree of node j, and BK_j, which represents the "blind key" for that node which will then get used to generate group keys for ancestor nodes. Let's look at a small picture of a system with four users:



Let's see how each key and blind key gets calculated:

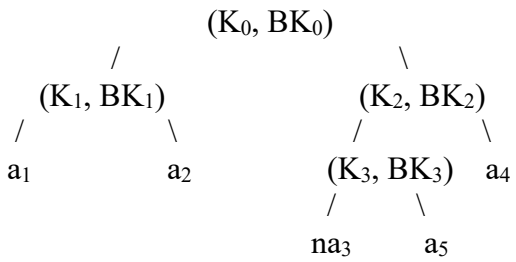
$K_1 = g^{a_1 a_2} \ (mod\ p) \ \rightarrow BK_1 = g^{K_1} \ (mod\ p)$
$K_2 = g^{a_3 a_4} \ (mod\ p) \ \rightarrow BK_2 = g^{K_2} \ (mod\ p)$
$K_0 = g^{K_1 K_2} \ (mod\ p) \ \rightarrow \ BK_0 = g^{K_0} \ (mod\ p)$, notice that since this is the root node, BK_0 technically never gets used.

Thus, K_1 is the shared key for users with secret keys a_1 and a_2.
K_2 is the shared key for users with secret keys a_3 and a_4.
K_0 is the shared key for all four users.

Note: We use the blind key as the piece of information that is publicly sent to the sibling node so that the new key for the parent node can be calculated.

Adding a New User

The system does not have to always be a perfect binary tree, Let's consider adding a new user with secret key a_5. Here is how the tree structure could change in this case:



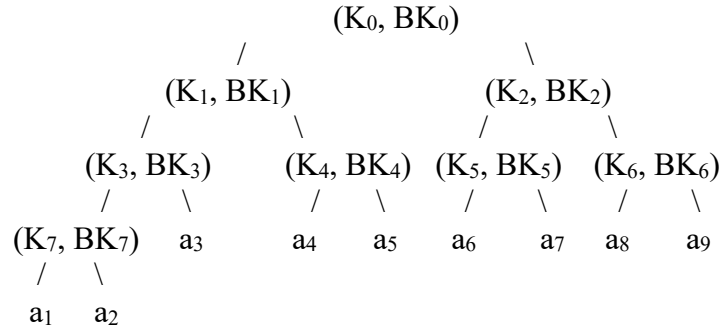
In this case, we just have to recalculate all keys up the ancestral path. The user with which we are splitting the new user will have to choose a new secret key, which is designated as na_3. Here are the new calculations:

$K_3 = g^{na_3 a_5} \ (mod\ p) \rightarrow BK_3 = g^{K_3} \ (mod\ p)$
$K_2 = g^{K_3 a_4} \ (mod\ p) \ \rightarrow BK_2 = g^{K_2} \ (mod\ p)$
$K_0 = g^{K_1 K_2} \ (mod\ p) \ \rightarrow BK_0 = g^{K_0} \ (mod\ p)$

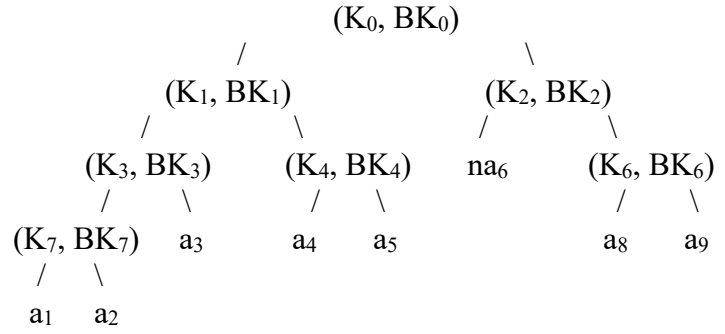
Basically, the new user affects every ancestor key, thus, we just have to update, in order, from bottom to top of the tree, each ancestor's key and blind key, going up the tree. In practice, this is likely done via recursion and as the function calls pop off the stack, the new keys are calculated in this order. For large trees, if there are n users, the number of updates, so long as we keep the tree reasonably balanced will be O(lg n), since the average height of a randomly composed binary tree of n elements is O(lg n).

Deleting a User

Deleting is similar to insertion. When a member is deleted, then changes must be made up the ancestral path. Since each user is always a leaf node, the cases for delete are not as complicated as deletion from a regular binary search tree. Let's consider an example:



Let's say the user with secret key a_7 leaves the group. This means that the node with K_5 will disappear and be replaced by a_6. But, because the groups have changed, we require that the user with secret key a_6 change it to na_6. Here is the new picture and the ensuing key changes:

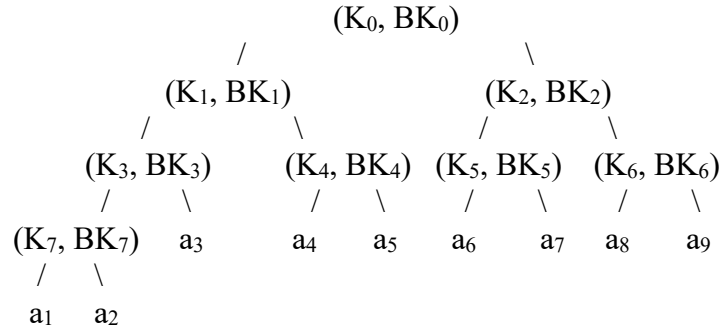


$K_2 = g^{K_6 na_6} \ (mod\ p) \rightarrow BK_2 = g^{K_2} \ (mod\ p)$

$K_0 = g^{K_1 K_2} \ (mod\ p) \ \rightarrow BK_0 = g^{K_0} \ (mod\ p)$

Small Example with Numbers

For simplicity, let's use p = 29, and g = 2. Let's use the picture from our previous example:



Let's pick the following secret keys for each user:

$a_1 = 23$, $a_2 = 11$, $a_3 = 17$, $a_4 = 9$, $a_5 = 16$, $a_6 = 18$, $a_7 = 24$, $a_8 = 5$, $a_9 = 9$

Here are the calculations for each of the keys and blind keys

$K_7 = 2^{23(11)} \bmod 29 = 2$ (note that 253 is equivalent to 1 mod 28…)
$BK_7 = 2^2 \bmod 29 = 4$

$K_4 = 2^{9(16)} \bmod 29 = 16$
$BK_4 = 2^{16} \bmod 29 = 25$

$K_5 = 2^{18(24)} \bmod 29 = 7$
$BK_5 = 2^7 \bmod 29 = 12$

$K_6 = 2^{5(9)} \bmod 29 = 21$
$BK_5 = 2^{21} \bmod 29 = 17$

$K_3 = (BK_7)^{a3} \bmod 29 = 4^{17} \bmod 29 = 6$
$= g^{(K7a3)} \bmod 29 = 2^{2(17)} \bmod 29 = 6$, showing that we can calculate this 2 ways by hand.
$BK_3 = 2^6 \bmod 29 = 6$ (note this is a coincidence)

$K_2 = (BK_5)^{K6} \bmod 29 = 12^{21} \bmod 29 = 12$
$= g^{K5K6} \bmod 29 = 2^{7(21)} \bmod 29 = 12$
$BK_2 = 2^{12} \bmod 29 = 7$

$K_1 = (BK_3)^{K4} \bmod 29 = 6^{16} \bmod 29 = 7$
$= g^{K3K4} \bmod 29 = 2^{6(16)} \bmod 29 = 7$
$BK_1 = 2^7 \bmod 29 = 12$

$K_0 = (BK_1)^{K2} \bmod p = 12^{12} \bmod 29 = 1$

As you can see with small numbers, lots of coincidences can happen, but hopefully this explains the process clearly.