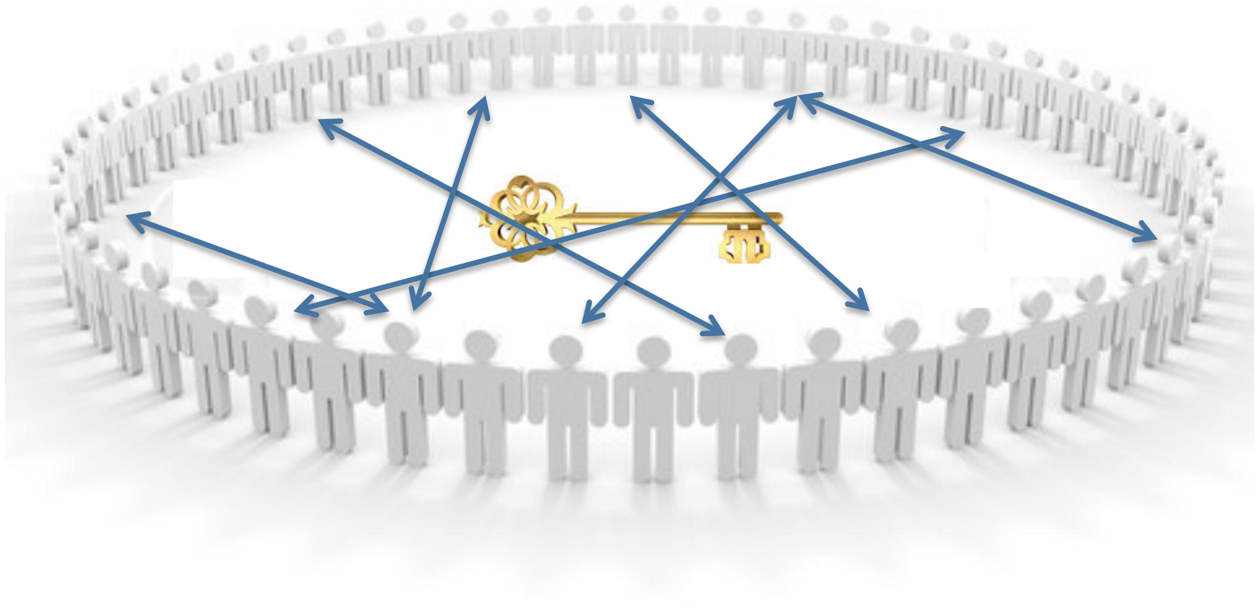
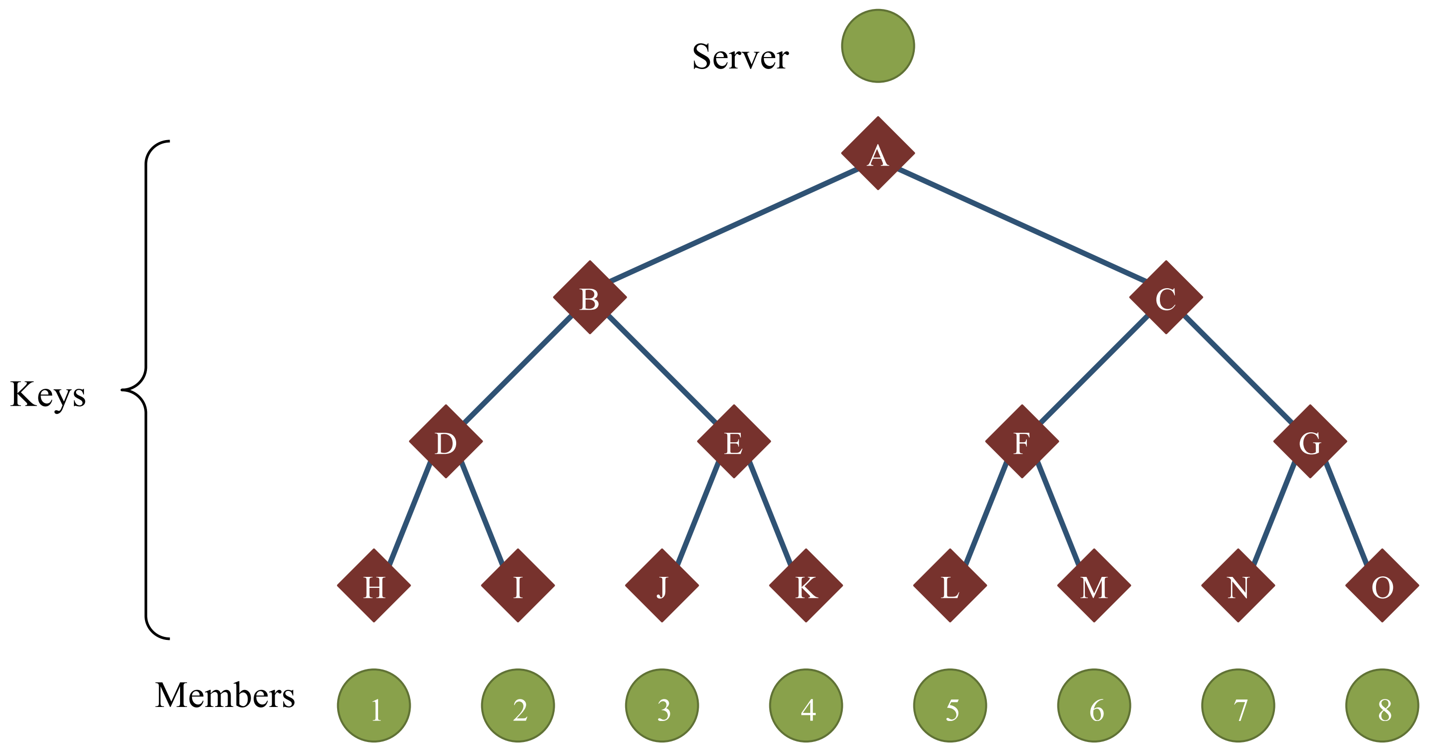
Group Key Distribution





This tree represents an efficient way to distribute keys to the members of a group if there is only one sender. If there are n members, then each member has to store $\log(n)$ keys, and the bandwidth will be at most $r \log\left(\frac{n}{r}\right)$, where r represents the number of revoked members.

STORAGE: $\log(n)$

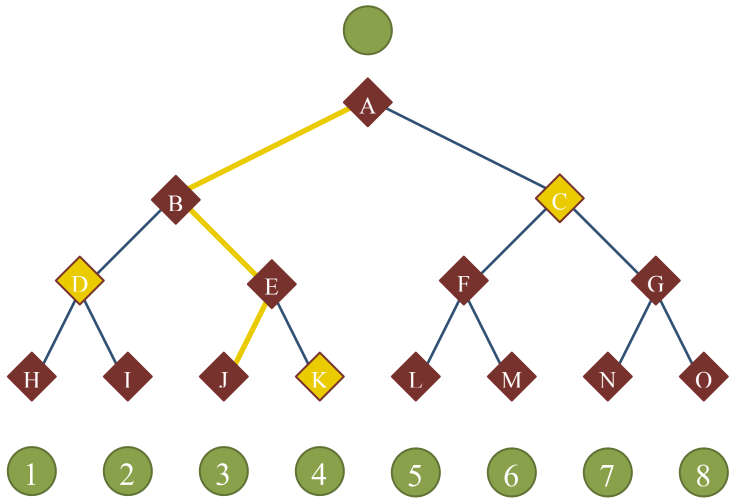
BANDWIDTH: $r \log\left(\frac{n}{r}\right)$

How Does This Scheme Work?

Each member is storing only the keys on the path from itself to the root. In order for a given member to receive a message from the server, the server must have used one of the keys in this member's path.

Example:

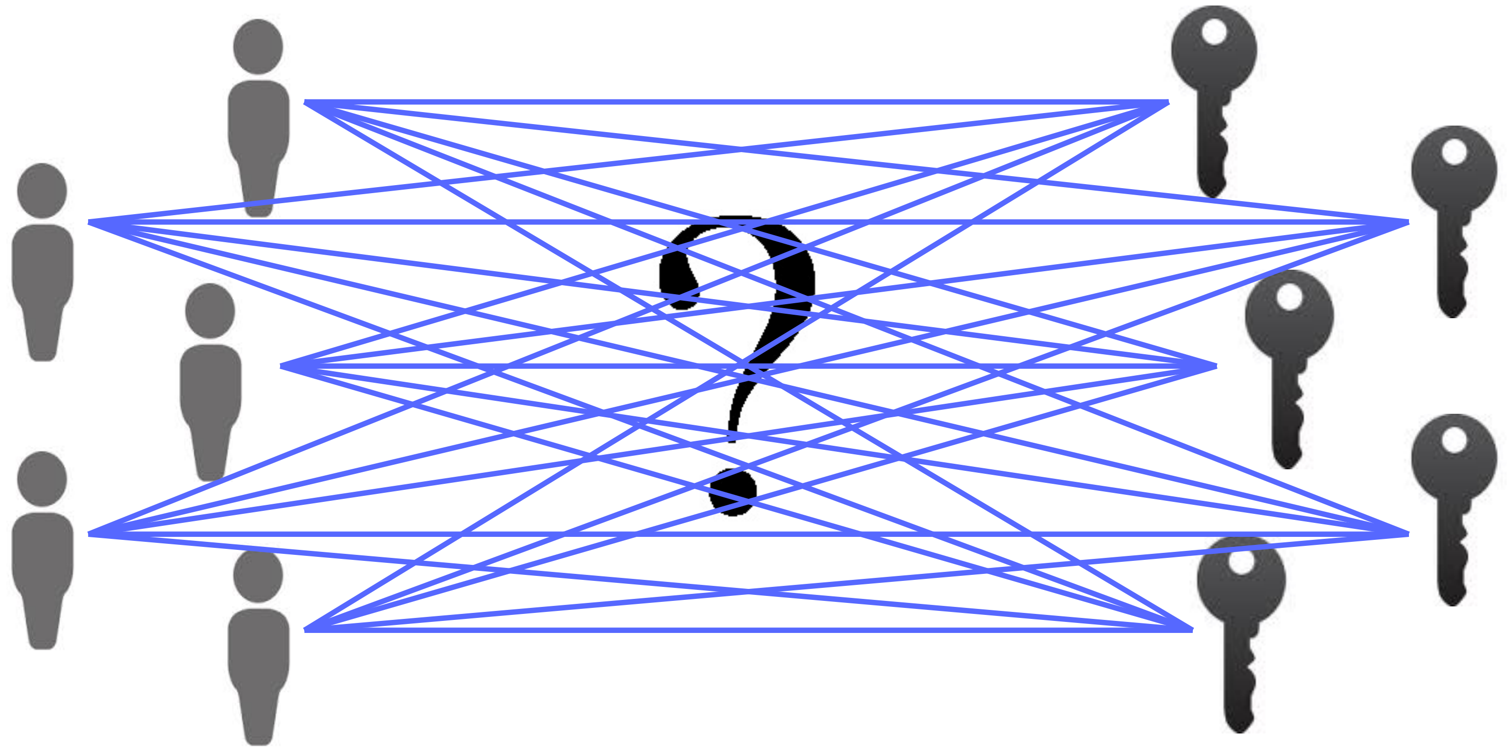
The server wants to send a message to all members except member 3. The server would identify the path from member 3 to the root, and then select the key at each level that is the sibling to the key in the path. Thus the server would select keys C, D, and K, which would allow every member besides 3 to decrypt the message.



The Problem

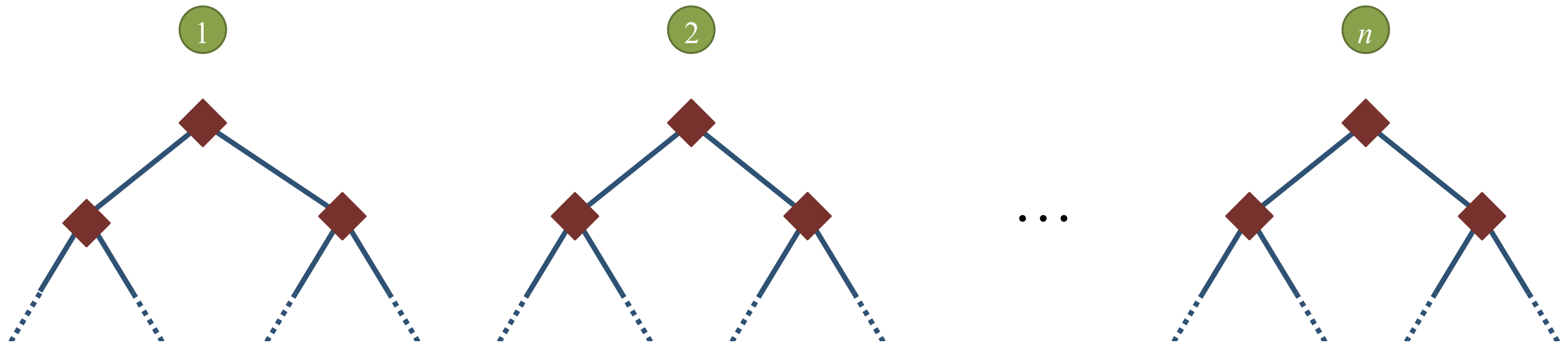
How should keys be distributed so that any member could send a message to any subset of the other members?

This problem has many possible answers, but the goal is to find a way to assign keys that optimizes the number of keys each member has to store, as well as the bandwidth for the sender.



The Trivial Solution

Generalize the scheme for one server by replicating it with each member acting as the server independently.



This solution would keep the bandwidth at $r \log\left(\frac{n}{r}\right)$, but each member would then have to store the $2(n-1)-1$ keys from its own tree, as well as $\log(n-1)$ for each of the other $(n-1)$ trees, giving a total storage of $(n-1)\log(n-1)+2(n-1)-1$, or $O(n \log(n))$.

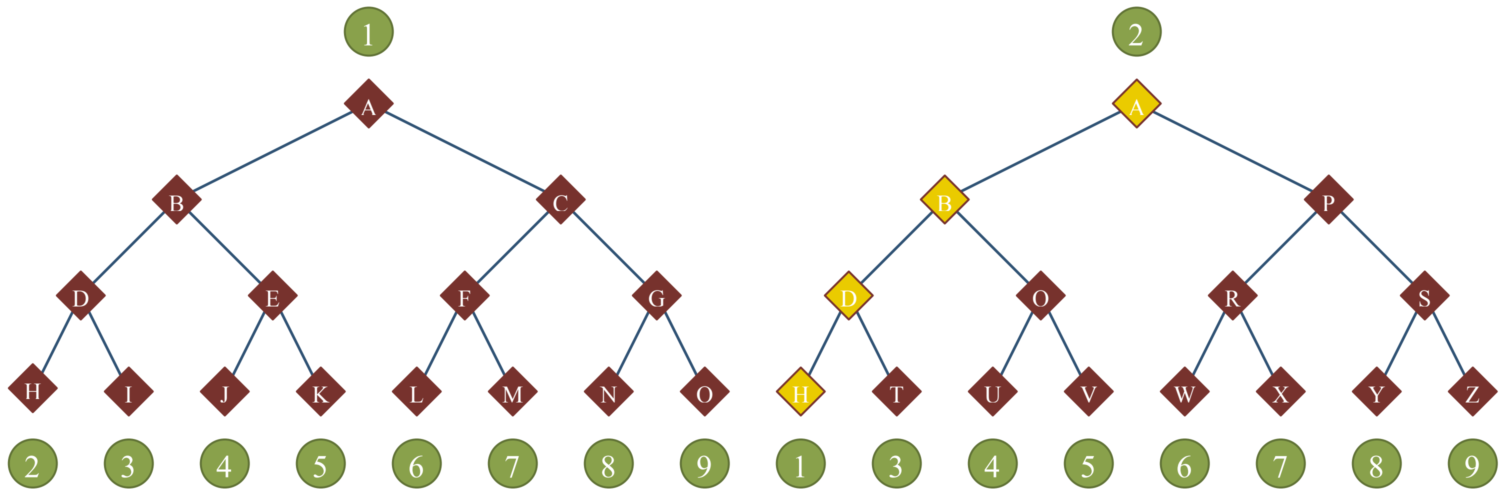
The problem then becomes: Is it possible to improve upon storage $O(n \log(n))$ and bandwidth $O(r \log\left(\frac{n}{r}\right))$?

Possible Approaches to the Problem

- Remove duplicate keys to improve storage
- Redefine key selection algorithm to improve bandwidth
- Assign keys differently to improve storage

Removing Duplicate Keys

In the trivial approach, the resulting set of trees will contain many keys that correspond to the same set of members. For example, in a group of 9 users, member 1's server tree and member 2's server tree both have keys for $\{1, 2\}$, $\{1, 2, 3\}$, $\{1, 2, 3, 4, 5\}$, and $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.



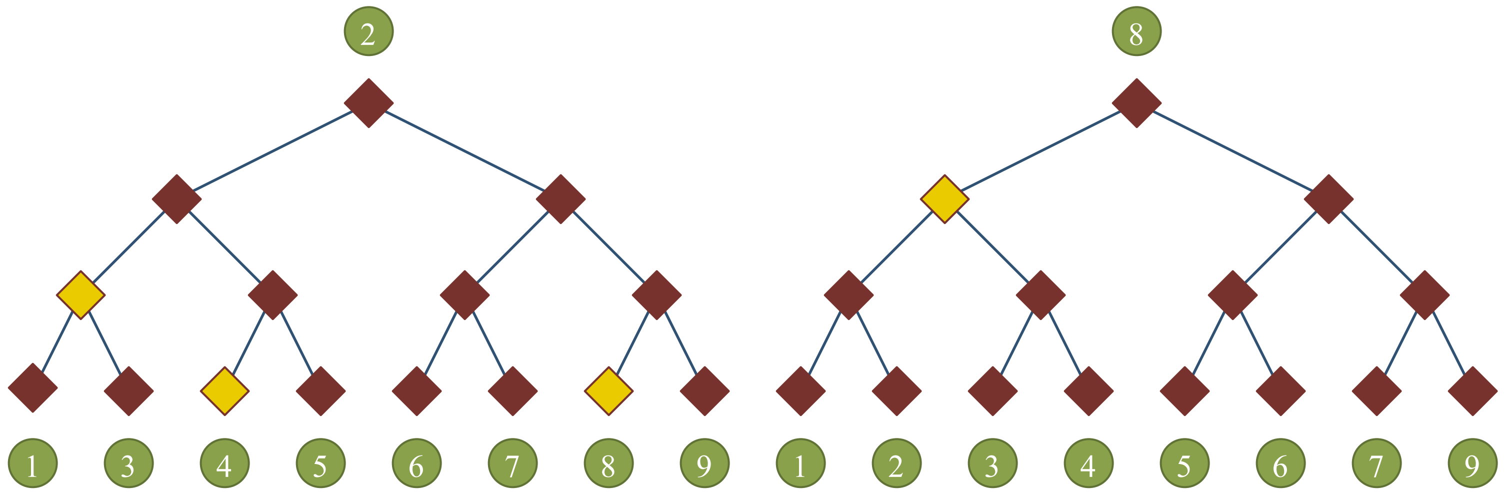
While this does reduce the number of keys each member needs, the storage is still $O(n \log(n))$.

Sender Key Selection

When all members have their own tree of keys, each member is storing more keys than just those in its own tree. Thus when a member wants to send a message, it would be useful to consider all the keys it has, rather than just those within its own tree.

Example:

In a system with 9 members, user 2 wants to send a message to users {1, 3, 4, 8}. If the server used only its own tree, it would need to use 3 keys to accomplish this. If, on the other hand, the server took advantage of the keys it is storing from the other members' trees, it would need only the one key that is held by {1, 2, 3, 4, 8}.



Designing a New Scheme

The binary tree of keys is efficient for a single server, but when every member needs to be a sender, there may be a better scheme for organizing keys. Some other system of assigning keys to subsets of users might minimize storage and bandwidth. The only rule is that every member must be able to send to any subset of the other members.

Storage $< O\,(n \log(n))$?

Bandwidth $< O\,(r \log\left(\frac{n}{r}\right))$?