well as other information that is vital to a secure link. Bob's machine responds by selecting the encryption algorithm and other features it can use from Alice's list. If there is a common set of methods between the two machines, then the protocol continues. The next two messages follow the Diffie–Hellman procedure. Alice's machine sends her part of the Diffie–Hellman secret plus a random value. Bob's machine responds with his part of the Diffie–Hellman secret and a random value. Now, both sides have a common secret key. Finally, Alice's machine sends its identity information and Bob's machine responds with its identity information.

9.1.2 Group Keys

Diffie–Hellman works fine for pairs of users, but there is a need for larger groups of users to securely establish a known key as well. Sometimes this situation arises when someone wants to broadcast a message to several different users. This is called a multicast, or one-to-many, communication link and will be covered in the next section. Other times, this situation arises when a group of users needs to carry on a secure discussion in a many-to-many communication link. In this case, every sender is also a receiver, and the process is called *peer-group communication.*

DH can be generalized to work in peer-group mode. For example, say Chris wants to join in with Bob and Alice. They need a common key, so they elect to use DH. All three agree on common values g and n. Alice begins by selecting a random number, a, and sends Bob the value g^a mod n. Bob selects his random number, b, and sends Chris the value g^b mod n to send to Alice. Finally, Chris selects a random number, c, and sends Alice the value g^c mod n. This completes the first step. Now Alice sends Bob the value g^{ac} mod n, Bob sends Chris the value g^{ab} mod n, and Chris sends Alice the value g^{bc} mod n. This completes the second step. For the third and final step, each one calculates the common key. Alice uses her random number and the final value Chris sent her to calculate $K = (g^{bc})^a = g^{abc}$ mod n. Bob uses his random number and the final value Alice sent him to calculate $K = (g^{ac})^b = g^{abc}$ mod n. Chris uses his random number and the final value Bob sent him to calculate $K = (g^{ab})^c = g^{abc}$ mod n. They all have the same key, but it is never transmitted directly, so it remains secure. Now they can establish a multicast or peer-group communication linkage. This process is illustrated in Figure 9.4.

While this offers Bob and Alice a straightforward generalization of the standard DH process, it does pose some problems. For example, if someone wants to join their group,

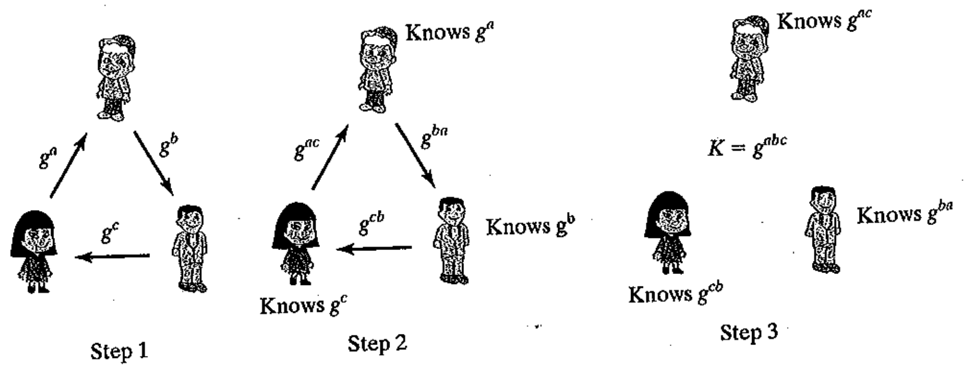

Figure 9.4: A group Diffie–Hellman process

they have to go through these steps again. Not only that, because there are four users involved, they have to go though four steps instead of three. If someone leaves the group, then the key assignment process must begin again as well. In short, this process does not offer the degree of flexibility required for large and variable-size groups.

Another, somewhat obvious, approach is to appoint a member of the group to be the group controller (GC). The GC shares a different key encryption key (KEK) with every member of the group. All the group members ($M_1, M_2, \ldots, M_n$) share a single traffic encryption key (TEK), which is used to send encrypted, secure messages to each other. This model looks something like a star as shown in Figure 9.5.

When a new member joins the group, the GC sets up a KEK for her and determines a new TEK. The TEK is sent to each member, encrypted using the member's individual KEK. If a member leaves the group, the GC determines a new TEK and sends it to the remaining member's encrypted, of course, in each member's KEK. This method does not scale well. A group of size n requires n key encryption keys and one traffic encryption key, or $n + 1$ keys overall.

Overall, a good group key-management system should not only provide an efficient method to add or drop members, it should also provide four forms of security:

1. Group key security—it should not be possible for an outsider to discover the key.
2. Backward security—it should not be possible to determine previous group keys from knowledge of a subset of the current set of group keys.
3. Forward security—it should not be possible to determine any future keys from knowledge of a subset of the current set of group keys.
4. Key independence—it should not be possible to determine any group key from knowledge of a subset of the current set of group keys.

Kim, Perrig, and Tsudik presented a key-management scheme at a 2000 Association for Computing Machinery (ACM) conference that offered better scaling and a more formal procedure for joining and exiting a group. It is called a tree-based group Diffie–Hellman protocol (TGDH). The relationship between keys is represented in a key tree such as the one shown in Figure 9.6 for a nine-member group. Each key box in the figure contains two keys: a secret key, K_i, and public key (called a blinded key), BK_i. The root key, K_0, is the group key. Each member has access to all the blinded keys and to all the secret keys in the path from the root node down to the member. For the example, M_4 knows secret keys K_4, K_1, and K_0, as well as all the public keys.

To start the process of creating the keys, all members agree on public values for g and p as in the DH protocol. Then each member will select a secret random number. (Say, member M_i selects a_i.) If two members are children of the same key node in the

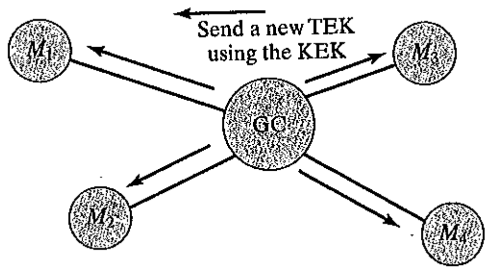

Figure 9.5: A star-shaped group key-management system

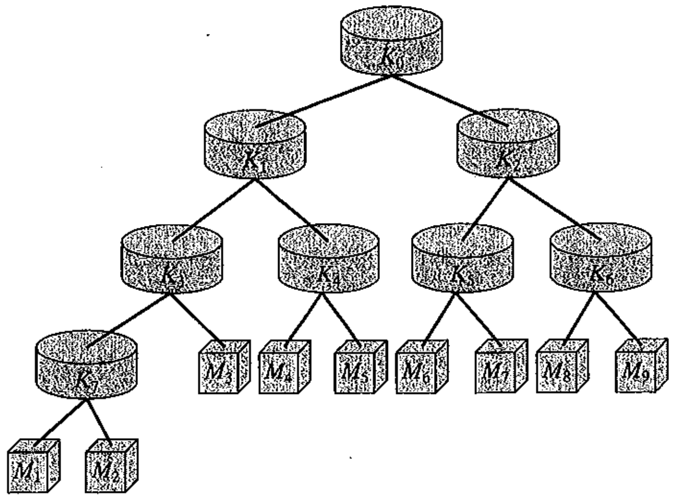

Figure 9.6: An example of a TGDH system

tree (as are M_1 and M_2), they will follow the standard DH protocol and create a shared secret key based on their individual random numbers. In the example (all values are reduced mod p),

$$M_1 \text{ sends } M_2 \text{ the value } g^{a1}$$
$$M_2 \text{ sends } M_1 \text{ the value } g^{a2}$$

Using these values, they calculate the secret key value

$$K_7 = g^{a_1 a_2}$$

and the public blinded key value

$$BK_7 = g^{K_7} = g^{g^{a_1 a_2}}.$$

The keys at each stage of the example are shown in Figure 9.7. The two children of each node establish the keys for the parent node using DH. For node K_3, the process looks like this:

$$M_3 \text{ sends } k_3 \text{ the value } g^{a3}$$
$$k_3 \text{ sends } M_3 \text{ the value } g^{k7}$$

The result is

$$K_3 = g^{a_3 K_7} \quad \text{and} \quad BK_3 = g^{K_3}.$$

Since every member knows all the secret keys in the path from their location up to the root node, every member knows the group key. They also have knowledge of intermediate keys, which could be used to form secure subgroups. For the example, M_1, M_2, and M_3 are the only members who know K_3, so they could use that key for secure communications among themselves.

If a member leaves the group, some of the keys have to be changed so old members cannot use knowledge of the prior keys to continue to eavesdrop on the group. In this case, a special member is appointed as the sponsor. The task of the sponsor is to broadcast the

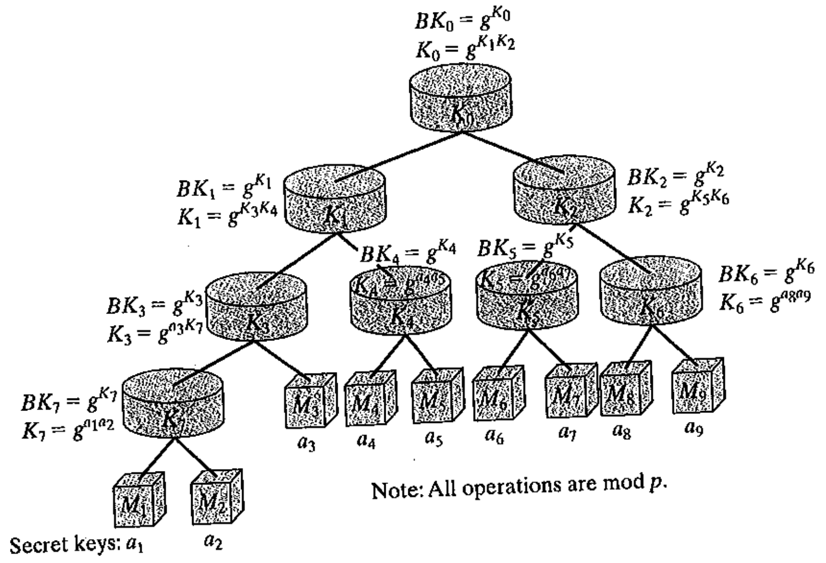

Figure 9.7: Keys in an example TGDH system

updated blinded keys so that the entire group can change their keys if necessary. For our example, M_7 leaves the group so M_6 becomes the sponsor. The two secret keys, K_0 and K_2, must be changed, since these are known to M_7. So M_6 will select a new random number, na_6, which will be used to update K_2. M_6 will broadcast the new BK_2 to every member so that M_8 and M_9 can update both K_0 and K_2 while the other members update just the group key K_0. The required changes are shown in Figure 9.8. Notice that only a few keys must be changed—this is really a very efficient method.

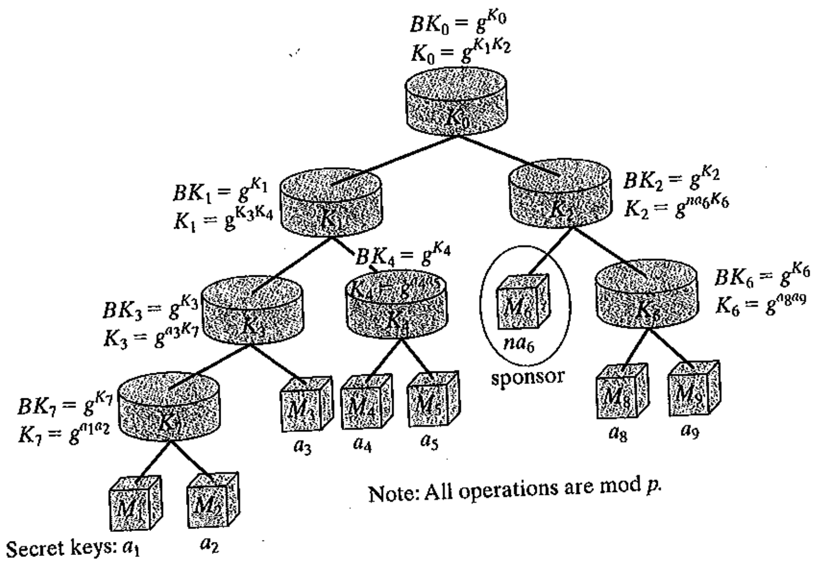

Figure 9.8: Rekeying when a member leaves

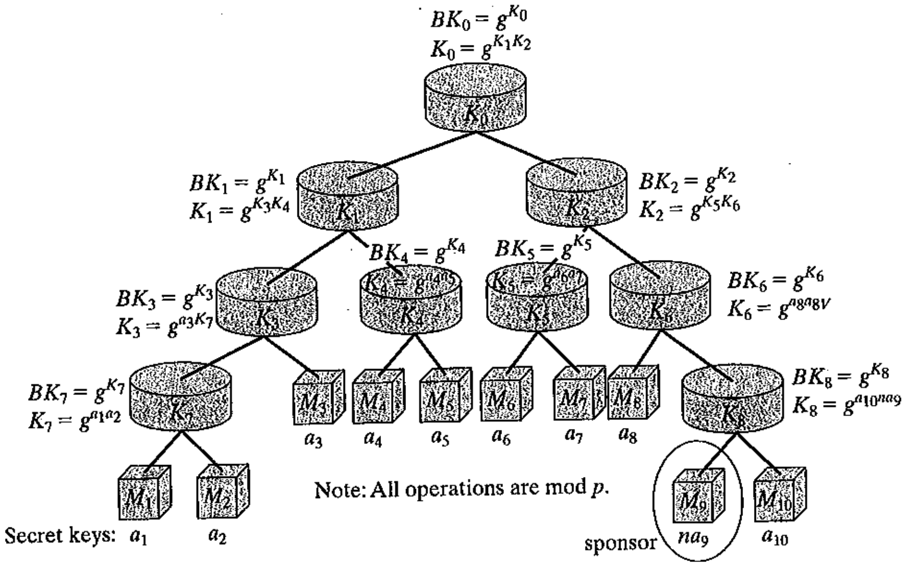

Figure 9.9: Rekeying when a member is added

Adding a new member to the group is also easy. As before, a sponsor is appointed who will update the blinded keys. For example, if M_{10} becomes a new member as shown in Figure 9.9, M_9 might be appointed the sponsor. In that case, M_9 would select a new random number, na_9, and use it to create K_8 and BK_8. This information is passed up the tree to create a new group key. Notice that, when M_{10} is added, he is not given any information that could reveal the prior keys. Every secret key in the path from M_{10} to the root is changed.

9.1.2 Broadcast Encryption

Up to this point, the assumption has been that multiple parties communicate with each other. They can share information back and forth in order to set up a key. But what happens when that is not the case? How can a key be established when one party can only transmit and another can only receive? This may seem like a unique and artificial condition, but it is actually quite common. Consider the operation of pay TV. Home viewers purchase a descrambling box and attach it to their TV. This box will accept an encrypted TV signal and decrypt it for viewing. How does the transmitting authority know which key to use to encipher the signal? How does the home box know the key for deciphering? The process of making this all work is called *broadcast encryption.*

The broadcast encryption problem is further complicated by the fact that it is usually only a subset of the users that are authorized to have access to a specific message. In the TV example, only a subset of the box owners may request a particular pay-per-view program, so only those owners should be able to decipher the TV signal. All other box owners should not be able to view the program for free. This is the case for those viewers who want to watch "The Sun Sets at Noon" as shown in Figure 9.10.

There are several methods of broadcast encryption that have been suggested and utilized. One method would be to assign each TV box its own private key and then send out a customized signal encrypted with each private key to each box. With millions of home users, this would require that the same TV signal be encrypted and transmitted millions of times. There must be a better way.