Establishment of Conference Keys in Heterogeneous Networks

Wade Trappe, Yuke Wang, K. J. Ray Liu
Department of Electrical and Computer Engineering, University of Maryland

***Abstract*— In order to secure communication amongst members of a conference, a secret shared by all group members must be established. The Diffie-Hellman problem is often the basis for generating keys in two-party communication, and can also be used to establish conference keys. In heterogeneous networks, many conferences have participants of varying computational power and resources. Most conference keying schemes do not address this concern and place the same burden upon less-powerful clients as more-powerful ones. The establishment of conference keys should try to minimize the burden placed on more resource-limited users while ensuring that the entire group can establish the group secret. In this paper, we present a scheme for establishing a conference key using the two-party Diffie-Hellman scheme. The scheme is hierarchical, forming subgroup keys for successively larger subgroups en route to establishing the group key. A full, binary tree called the conference tree governs the order in which subgroup keys are formed. Key establishment schemes that consider users with varying costs or budgets are designed by appropriately choosing the conference tree. The tree that minimizes the total group cost is produced via the Huffman algorithm. A criterion is presented for the existence of a conference tree when users have varying budgets, and a greedy algorithm is presented that minimizes the total length of the conference tree under budget constraints.**

***Index Terms*— Conference key, Diffie-Hellman, Huffman Coding**

I. Introduction

The advancement of communication technology is leading to a future where group-based applications will become a reality. Many applications will require that the communication amongst group members be protected from unwanted eavesdroppers. Corporate conferences, with members from different parts of the world, might contain industrial secrets that are in the best interests of the corporation to keep unknown to rivals. In order to protect the communication traffic, the information must be encrypted, which requires that the privileged parties share an encryption and decryption key. Key distribution is accomplished either by using a centralized entity that is responsible for distributing keys to users, or by contributory protocols where legitimate members exchange information that they can use to agree upon a key.

In the centralized approach to group key establishment, either a group leader or a trusted third party is responsible for the generation and distribution of keying material. The problem of centralized group key distribution has seen considerable attention recently in the literature [1–3]. In many cases, however, it is not possible to have a third party arbitrate the establishment of a group key. This might occur in applications where group members do not explicitly trust a single entity, or no member has the resources to maintain, generate and distribute information by himself. In these cases, the group members make independent contributions to the formation of the group key, and the process of forming the key is called *key agreement*.

The classic example of a two-party key agreement protocol is the Diffie-Hellman key exchange scheme [4]. Establishing a shared secret amongst more than two parties is significantly more complicated than the two-party case. Several researchers [5–8] have studied the problem of establishing a *Diffie-Hellman like* conference key.

Typically, these conference key establishment schemes seek to minimize either the amount of rounds needed in establishing the group key, or the size of the message. Many applications, however, will involve a heterogeneous clientele in which group members will have different computational capabilities, pricing plans, and bandwidth resources. For these applications, minimizing the total bandwidth or amount of rounds might not be an appropriate metric. Instead, one should aim to minimize a cost function that incorporates the different costs of each user. Additionally, users might have resource constraints imposed upon them, in which case the key generation procedure must decide whether it is feasible to generate a key and determine a procedure for generating the group key while minimizing the total cost subject to resource budget constraints.

In this paper we study the problem of establishing Diffie-Hellman conference keys for applications with clients of differing cost profiles. Using the two-party Diffie-Hellman protocol as the basic building block, we can establish a group key by forming intermediate keys for successively larger subgroups. A full, binary tree that we call the conference tree governs the order in which subgroup keys are formed. We then consider two problems: first, designing the conference tree such that the total group cost is minimized, and second, addressing the issue of forming a group key when users have varying budget constraints.

II. Group DH Overview

In many communication scenarios, there might not be a centralized entity that is available to manage and distribute the keys

Wade Trappe is with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742 USA, E-mail: wxt@eng.umd.edu

Yuke Wang is with the Department of Computer Science, University of Texas at Dallas, Dallas, TX USA, E-mail: yuke@UTDALLAS.EDU

K.J. Ray Liu is with the Department of Electrical and Computer Engineering & Institute for Systems Research, University of Maryland, College Park, MD 20742 USA, E-mail: kjrliu@eng.umd.edu

0-7803-7400-2/02/$17.00 © 2002 IEEE

to group members. In these cases, the members must interactively establish a shared secret by exchanging messages.

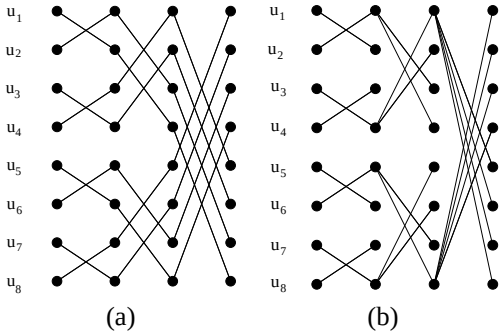
One of the most popular key establishment procedures for forming a shared key to secure a two-party communication is the Diffie-Hellman (DH) scheme [4]. In the basic DH scheme, the operations take place in an Abelian group G, typically chosen to be Z_p (the integers mod a prime p), or the points on an elliptic curve. For consistency, we shall develop our results for the group Z_p. A group element g is chosen such that g generates a suitably large subgroup of Z_p. Both party A and party B choose a private secret $\alpha_j \in Z_p^*$, where $j \in \{A, B\}$ and Z_p^* denotes the non-zero elements of Z_p. They each calculate $y_j = g^{\alpha_j}$ and exchange y_j with each other. Party A then calculates the key K via $K = (g^{\alpha_B})^{\alpha_A} = g^{\alpha_B \alpha_A}$, and similarly for party B. The Diffie-Hellman problem (for someone who intercepts g^{α_A} and g^{α_B} to determine $g^{\alpha_A \alpha_B}$) is thought to be computationally intractable for large subgroups generated by g.

Schemes that employ the Diffie-Hellman problem for establishing a secret amongst more than two parties have been described recently in [5–8]. The amount of messages sent and received, as well as the amount of bandwidth consumed are important measures of a protocol's efficiency. Another important measure that arises is the amount of rounds that a protocol requires to establish a group secret. In [8], the communication complexity involved in establishing a group key is studied, and lower bounds are provided for the total number of messages exchanged, and the amount of rounds needed to establish the group key. They also present a key establishment scheme based upon a hypercube structure in which the amount of rounds needed to establish the key is logarithmic in the group size. A similar technique was proposed in [9], where the problem of group key establishment was examined in terms of signal flow graphs. The basic approach, called the *butterfly* scheme, had communication flow that was reminiscent of the butterfly diagrams of FFT calculations. In fact, the hypercube approach of [8] is similar to the butterfly scheme when the two-party DH scheme is used as the basic building block. We describe the butterfly scheme in the following section.

III. Conference Trees and Group Keys

In the butterfly scheme [9], the users form keys for small subgroups, and these subgroups act as single entities that form larger subgroups and establish new keys using the previous subgroup keys. The process repeats until the entire group has formed a key that was shared by all members. A description of the butterfly scheme for $n = 2^r$ members is provided using the DH scheme.

Initially, suppose each user u_j has a random secret integer $\alpha_j \in Z_p^*$. The n users are broken into pairs of users $u_j^1 = \{u_{2j-1}, u_{2j}\}$. The superscript in the notation denotes which round of pairings we are dealing with, while the subscript references the pair. We shall refer to the initial secrets that each user possesses as $x_j^0 = \alpha_j$. In the first round, the members of a pair exchange $g^{x_j^0}$. For example, u_1 sends $g^{x_1^0}$ to u_2, and u_2 sends $g^{x_2^0}$ to u_1. Then, the users u_{2j-1} and u_{2j} each calculate $x_j^1 = g^{x_{2j-1}^0 x_{2j}^0} = g^{\alpha_{2j-1}\alpha_{2j}}$. Observe that $x_j^1 \in Z_p^*$, and that both members of a pair have established a conventional DH key. We may now pair the pairs u_j^1 into a second level of pairs, e.g. $u_1^2 = \{u_1^1, u_2^1\}$, and more generally $u_j^2 = \{u_{2j-1}^1, u_{2j}^1\}$. Hence, the second level of pairings consists of 4 users in a pair. Each user from u_{2j-1}^1 has an associated member of u_{2j}^1 to whom they send $g^{x_{2j-1}^1}$ and similarly receive $g^{x_{2j}^1}$ from. Every member in u_j^2 can calculate $x_j^2 = g^{x_{2j-1}^1 x_{2j}^1}$. A third pairing, consisting of 8 users may be formed and a similar procedure carried out if needed. In general, $u_j^k = \{u_{2j-1}^{k-1}, u_{2j}^{k-1}\}$ and $x_j^k = g^{x_{2j-1}^{k-1} x_{2j}^{k-1}}$. Ultimately, the procedure works its way down to where there are only two intermediate values that can be combined to get the group secret.



(a) (b)

Fig. 1. The radix-2 butterfly scheme for establishing a group key for 8 users. (a) Without broadcasts, (b) Using broadcasts.

A trellis diagram depicting the communication flows between $n = 8$ users is depicted in Figure 1(a). It is not necessary that each user perform a communication during each round. In fact, such an operation is inefficient since many users are transmitting identical information. In networks, such as wireless networks, where broadcasting is available, the communication can be arranged so that one user broadcasts a message to multiple users. An example trellis for such an arrangement is depicted in Figure 1 (b).

The general form of the butterfly scheme uses the scheme of [5] as the basic building block, and provides a broad family of approaches whose total rounds were logarithmic in the group size. It can be shown that using the two-party DH scheme (which is a special case of the scheme of [5]), produces the group key in the fewest amount of rounds. Therefore, we shall use only the two-party DH scheme in constructing the group key.

The idea of forming successively larger subgroups in the butterfly scheme can be generalized. Consider three users u_1, u_2, and u_3 who have made public a generator $g \in Z_p^*$ and who have secret exponents $\alpha_j \in Z_p^*$. If u_2 and u_3 participate in a two-party DH scheme, they establish the shared secret $y = g^{\alpha_2 \alpha_3}$. This shared secret y can be used as the key for the subgroup $\{u_2, u_3\}$, and also as a new secret exponent for a two-party DH scheme between u_1 and the subgroup $\{u_2, u_3\}$. To do this, a representative of $\{u_2, u_3\}$ sends g^y to u_1 while u_1 sends g^{α_1} to

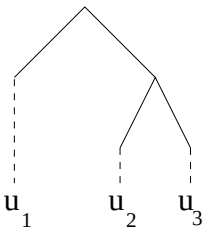


Fig. 2. The conference tree for $n = 3$ users, where users u_2 and u_3 form a subgroup.

both members of $\{u_2, u_3\}$. All three members can calculate the group secret $g^{\alpha_1 y} = g^{\alpha_1 g^{\alpha_2 \alpha_3}}$.

The tree depicted in Figure 2 describes the formation of the group key. Since the subgroup $\{u_2, u_3\}$ established a subgroup key before u_1 could participate, we represent the formation of this subgroup key by an internal node in a binary tree. Since u_1 was involved in the final stage of the procedure, the key formed is represented by the root node.

Any full binary tree with n terminal nodes can be used to describe a procedure for generating a group key for n users. We define a *conference tree* to be a full binary tree that describes the successive subgroups and intermediate keys that are formed en route to establishing the key for the entire group. The conference tree allows for great design flexibility. In the next section we shall use the concept of a conference tree to address the varying costs and limitations of the users.

Another useful application of the conference tree is that it allows for the specification of which subgroups can securely communicate amongst themselves since the internal nodes of the tree specify subgroup keys. As an example, consider the conference tree for the radix-2 butterfly scheme of [9] for $n = 8$ users, which is depicted in Figure 3. Observe that there is a node on the conference tree that is the grandparent of $\{u_1, u_2, u_3, u_4\}$ and hence there is a subgroup key that is shared by $\{u_1, u_2, u_3, u_4\}$. This subgroup can choose to use this key to communicate securely amongst themselves if desired.

If a subgroup U of users desires to communicate securely amongst themselves in addition to participating in the entire group conference, then they must simply assure that a node of the conference tree covers the users of U. When two subgroups are nested, e.g. $U_1 \subset U_2$, it is possible to design a conference tree where U_2 is covered by a node that is an ancestor of the node covering U_1. When the two subgroups are disjoint, i.e. $U_1 \cap U_2 = \emptyset$, the two subgroups can be placed on separate branches of the conference tree and the generation of their subgroup keys occurs independently. The conference tree does not allow for all possible hierarchies of subgroups, but instead allows for nested hierarchies of subgroups. For example, if $U_1 \cap U_2 \neq \emptyset$ and $U_1 \not\subset U_2$ or $U_2 \not\subset U_1$, then the users of $U_1 \cap U_2$ must participate in two separate subgroup key generation procedures. The conference tree structure only allows for subgroups of users to participate in one two-party DH procedure at a time. Generalizations of the tree, such as graphs, may allow for more arbitrary subgroup arrangements.

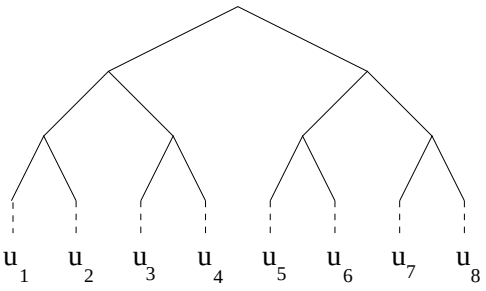


Fig. 3. The conference tree for the radix-2 butterfly scheme for establishing a group key for $n = 8$ users.

IV. Computational Considerations

In many application environments the users will have varying amounts of computational resources available. Low-power devices, such as wireless appliances, cannot be expected to expend the same amount of computational effort as a high-power device, such as a personal computer, when establishing a group secret. It is therefore important to study the problem of efficiently establishing a conference key while considering the varying user costs.

In this section we present methods for designing the conference tree used in establishing the group secret. In particular, we study two problems: minimizing the total cost in establishing a group key, and the feasibility of establishing the group key in the presence of budget constraints.

A. Minimizing Total Cost

First, assume that we have n users, and that each user u_j has a cost $w_j \in [1, B]$ associated with performing one two-party Diffie-Hellman protocol. For example, this cost might be related to the amount of battery power consumed. The value B is used to denote an upper bound placed on the cost that a user may have. Suppose we place the n users on a conference tree with n terminal nodes in such a manner that each user u_j has a length l_j from his terminal node to the root of the conference tree. Our goal is to minimize the total cost C of this tree $C = \sum w_j l_j$.

We first address the question of what is the minimum amount of total computation necessary for establishing the group key for n users. This problem can be addressed using coding theory. Define $p_j = w_j / (\sum_k w_k)$, then $\sum_j p_j l_j$ is just a scaling of $\sum_j w_j l_j$ by $W = \sum_k w_k$. If we define X to be a random variable with a probability mass function given by p_j then minimizing $\sum_j p_j l_j$ is equivalent to finding a code for X with lengths l_j that minimizes the average code length. With this observation, we may infer the following lower bound on the total cost for establishing a group key, which follows from the lower bound for expected codelength of an instantaneous binary code for X:

Lemma 1: Suppose that n users wish to establish a group secret and each user u_j has a cost w_j associated with per-

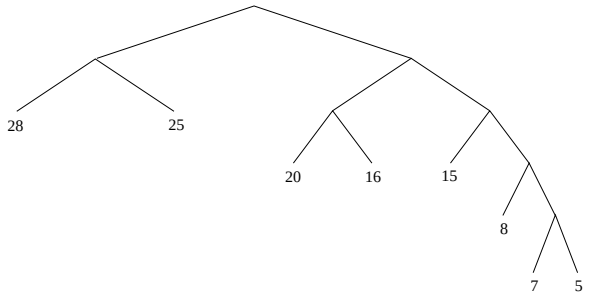


Fig. 4. Huffman example

forming one two-party Diffie-Hellman protocol. Then the total cost C of establishing the group secret satisfies the bound $-W \sum_j p_j \log_2 p_j \leq C$, where $p_j = w_j/W$.

The observation that efficiently establishing a group key is related to coding allows us to use procedures from coding theory to determine efficient conference trees. In particular, Huffman coding [10] produces the conference tree that minimizes the cost: if C^* is the cost of forming the group key using the Huffman tree, then the cost C' of using a different conference tree assignment will satisfy $C' \geq C^*$.

Since Huffman coding produces an optimal code, we know that the expected cost $\sum_j w_j l_j^*$ satisfies the bound $WH(p) \leq \sum_j w_j l_j^* < W\left(H(p)+1\right)$, where $H(p)$ is the entropy of the distribution p. Thus, the Huffman construction of the conference key tree has a total cost that is within W of the lower bound.

Consider the following example that demonstrates the advantage of using the Huffman algorithm for forming the conference tree when compared to using the full balanced tree that corresponds to the radix-2 butterfly scheme.

Example 2: Consider a group of 8 users with costs $w_1 = 28$, $w_2 = 25$, $w_3 = 20$, $w_4 = 16$, $w_5 = 15$, $w_6 = 8$, $w_7 = 7$, and $w_8 = 5$. The Huffman algorithm yields the tree depicted in Figure 4. The corresponding length vector is $l^* = (2, 2, 3, 3, 3, 4, 5, 5)$, and the total cost is 351. The total cost for a full balanced tree is 372.

We would like to be able to quantify the improvement that is available when one uses the Huffman code compared to the cost of using an arbitrary conference tree. For an arbitrary conference tree, suppose that the length for user u_j is l_j. If we define a probability distribution q by $q_j = 2^{-l_j}$, then the expected length under the probability p_j of the code with lengths l_j satisfies [10]

$$H(p) + D(p\|q) \leq \sum_{j=1}^{n} p_j l_j < H(p) + D(p\|q) + 1. \quad (1)$$

Here $D(p\|q)$ is the Kullback-Leibler distance between the two probability distributions p and q. The cost for using this tree is $C = W \sum p_j l_j$. We can combine the above bound with the bound for the cost of the optimal code $C^* < W(H(p)+1)$ to get

$$C - C^* > W\left(D(p\|q) - 1\right). \quad (2)$$

When $D(p\|q) > 1$ this bound is an improvement over the trivial bound $C - C^* \geq 0$.

Next, we consider the effect one user can have upon the computational cost of the remaining users. Recall that if the weights are ordered as $w_1 \geq w_2 \geq \cdots \geq w_n$ then the lengths of the Huffman code can be ordered as $l_1^* \leq l_2^* \leq \cdots \leq l_n^*$. Therefore, if user u_1 would like to adversely affect the lengths of the other users, he should announce as large of a weight as possible.

Suppose user u_1 publishes a weight $w_1 = B$. We now quantify the extra cost he places upon the other $n-1$ users. Define $\tilde{W} = \sum_{k=2}^{n} w_k$ and define the probability $q_j = w_j/\tilde{W}$ for $j \in \{2, 3, \cdots, n\}$, and $q_1 = 0$. Then q_j represents the probabilities that would be used in constructing a conference tree if user u_1 were not participating. Let l_j^* denote the optimal codelengths constructed using p_j, and $\tilde{l}_j^*$ be the optimal codelengths constructed using q_j.

We define $C^* = \sum_{j=1}^{n} w_j l_j^*$, $\tilde{C}^* = \sum_{j=2}^{n} w_j \tilde{l}_j^*$, and $C_{ex}^* = \sum_{j=2}^{n} w_j l_j^*$. We are interested in comparing C_{ex}^*, which is the total cost of the remaining $n-1$ users given the probabilities p_j which incorporate u_1's cost, with $\tilde{C}^*$, which is the total cost of the $n-1$ users $u_2, u_3, \cdots, u_n$ without considering u_1's announced cost.

First, since $\tilde{C}^*$ arises as the optimal code for the $n-1$ users with costs $w_2, w_3, \cdots, w_n$, we know $\tilde{C}^*$ minimizes costs of the form $\sum_{j=2}^{n} w_j l_j$. In particular, C_{ex}^* must satisfy:

$$C_{ex}^* = \sum_{j=2}^{n} w_j l_j^* \geq \sum_{j=2}^{n} w_j \tilde{l}_j^* = \tilde{C}^*. \quad (3)$$

We now derive an upper bound for C_{ex}^*. Observe that the code given by $\tilde{l}_j^*$ can be used to construct a code for p_j by taking $l_1 = 1$ and $l_j = \tilde{l}_j^* + 1$. The optimal code for the weights $w_1, w_2, \cdots, w_n$ must be better than this code, and hence

$$C^* \leq w_1 + \sum_{j=2}^{n} w_j(\tilde{l}_j^* + 1) = \tilde{C}^* + W. \quad (4)$$

Since $C_{ex}^* = C^* - w_1 l_1^*$, we have the following bound

$$C_{ex}^* \leq \tilde{C}^* + W - w_1 l_1^* \leq \tilde{C}^* + \tilde{W}. \quad (5)$$

Gathering the results together, we get the overall bound

$$\tilde{C}^* \leq C_{ex}^* \leq \tilde{C}^* + \tilde{W}. \quad (6)$$

This can be interpreted as meaning that user u_1 can, in the worst case, force the other $n-1$ users to spend an extra $\tilde{W}$ of resources. Combining this result with the bound for the optimal code for q produces

$$\tilde{W}H(q) \leq C_{ex}^* < \tilde{W}H(q) + 2\tilde{W}. \quad (7)$$

B. Budget Constraints

In many cases, the devices wishing to establish a conference key might have a limited budget to spend. The optimal conference key tree assignment that results from Huffman coding

might assign more computation to some users than they are capable of performing, while assigning less computation to other users than they are capable of performing. In these cases, rather than minimize the total cost, one wants to ensure that one can first establish the group key, and then consider reducing the total amount of computation as a secondary issue.

Suppose that user u_j publishes a budget b_j that describes the amount of two-party Diffie-Hellman key establishment protocols that he is willing to participate in when establishing the group key. Without loss of generality, we assume that the user's budgets b_j satisfy $b_j \leq b_k$ for $j < k$. Define the budget vector $b = (b_1, b_2, \cdots, b_n)$. The necessary conditions on the budget vector b for the existence of a conference key tree with lengths $l_j \leq b_j$ is provided by the Kraft Inequality [10]:

Lemma 3: Suppose that the budget vector $b = (b_1, b_2, \cdots, b_n)$. Then a conference key tree with lengths l_j exists that satisfies the budget constraint $l_j \leq b_j$ for all j if $\sum_{j=1}^{n} 2^{-b_j} \leq 1$.

A budget vector that satisfies the Kraft Inequality is said to be *feasible*. In the case that a budget assignment does not satisfy the Kraft Inequality, if we choose to drop a single member in hopes of having a feasible budget vector for the remaining users, then the best strategy is to drop the member with the lowest budget b_1.

Using the budget vector as the length vector does not necessarily lead to a full conference tree in which every node has two children. We must trim the budget vector in order to get a full tree. This can be accomplished by reducing elements of the length vector by amounts that do not violate the Kraft Inequality. The following lemma provides a useful approach to trimming the length vector assignment while still satisfying the Kraft Inequality.

Lemma 4: Suppose $b = (b_1, b_2, \cdots, b_n)$ strictly satisfies the Kraft Inequality, $\sum 2^{-b_j} < 1$, then the modified budget vector c defined by $c = (b_1, b_2, \cdots, b_{n-1}, b_n - 1)$ satisfies the Kraft Inequality $\sum 2^{-c_j} \leq 1$.

A consequence of this is that if we subtract 1 from one of the b_j then choosing the largest b_j least affects $\sum 2^{-b_j}$. The following algorithm uses this idea. It starts with an admissible budget vector b, initializes the length vector $l = b$, and produces a length assignment $l = (l_1, l_2, \cdots, l_n)$ satisfying $l_j \leq b_j$ such that $\sum 2^{-l_j} = 1$ and $\sum l_j$ is minimized over all length vectors c satisfying $\sum 2^{-c_j} \leq 1$. It can be shown that the greedy strategy employed in the algorithm leads to the optimal length vector.

As an example of the algorithm, suppose $n = 8$ and that the initial budget is $b = (1, 3, 3, 4, 5, 5, 6, 8)$. This budget vector is feasible and performing the algorithm gives the final assignment $l = (1, 3, 3, 4, 4, 4, 5, 5)$.

Data : A length vector l satisfying $\sum 2^{-l_j} \leq 1$.
while $\sum 2^{-l_j} < 1$ **do**
    $j = \arg\max\{l_k\}$;
    $l_j = l_j - 1$;
end

V. Conclusions

In this paper we have studied the problem of establishing conference keys when the users have different cost profiles or different budget constraints. It was shown that the users can use the two-party Diffie-Hellman protocol as a primitive for building a procedure that produces a group key. This procedure involves subgroups establishing keys that are then used to establish keys for successively larger subgroups. A binary tree, called the conference tree, governs the order in which the subgroups combine and this observation allows for determining procedures using Huffman coding that establish the group key and minimize the total user cost. Bounds were provided that describe the amount of improvement that can be achieved when compared to an arbitrary conference tree. We also studied the effect that one user can have upon the cost of the other $n - 1$ users. It was shown that the amount of additional cost that one user can impose upon the other users is bounded between the optimal cost for the $n-1$ users to establish the group key, and the optimal cost for the $n - 1$ users plus the sum of the $n - 1$ users' costs. We then addressed the problem of establishing a group secret when the users have limitations on the amount of Diffie-Hellman rounds they may participate in. We observed that in order for the group to establish a key, it is necessary that the budget vector satisfy the Kraft Inequality. We then presented an algorithm that trimmed the budget vector to produce a length assignment that satisfies the budget constraint and minimizes the unweighted total length of the conference tree.

References

[1] R. Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas, "Multicast security: a taxonomy and some efficient constructions," in *IEEE INFOCOM'99*, 1999, pp. 708 –716.

[2] C. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," *IEEE/ACM Trans. on Networking*, vol. 8, pp. 16–30, Feb. 2000.

[3] W. Trappe, J. Song, R. Poovendran, and K.J.R. Liu, "A dynamic key distribution scheme using data embedding for secure multimedia multicast," *Submitted to IEEE Trans. on Multimedia*, 2000.

[4] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, vol. 22, pp. 644–654, 1976.

[5] I. Ingemarsson, D. Tang, and C. Wong, "A conference key distribution system," *IEEE Transactions on Information Theory*, vol. 28, pp. 714–720, September 1982.

[6] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution scheme," *Advances in Cryptology- Eurocrypt*, pp. 275–286, 1994.

[7] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman key distribution extended to group communication," in *Proc. 3rd ACM Conf. on Computer Commun. Security*, 1996, pp. 31–37.

[8] K. Becker and U. Wille, "Communication complexity of group key distribution," in *5th ACM Conf. on Computer Commun. Security*, 1998, pp. 1–6.

[9] W. Trappe, Y. Wang, and K.J.R. Liu, "Group key agreement using divide-and-conquer strategies," in *Conference on Information Sciences and Systems, The John's Hopkins University*, March 2001.

[10] T. Cover and J. Thomas, *Elements of Information Theory*, John Wiley and Sons, 1991.