Resource-Aware Conference Key Establishment for Heterogeneous Networks

Wade Trappe, *Member, IEEE*, Yuke Wang, and K. J. Ray Liu, *Fellow, IEEE*

***Abstract*—The Diffie–Hellman problem is often the basis for establishing conference keys. In heterogeneous networks, many conferences have participants of varying resources, yet 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 minimize the burden placed on resource-limited users while ensuring that the entire group can establish the key. In this paper, we present a hierarchical conference keying scheme that forms subgroup keys for successively larger subgroups en route to establishing the group key. A 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 built by appropriately designing the conference tree. We then examine the scenario where users have both varying costs and budget constraints. A greedy algorithm is presented that achieves near-optimal performance, and requires significantly less computational effort than finding the optimal solution. We provide a comparison of the total cost of tree-based conference keying schemes against several existing schemes, and introduce a new performance criterion, the probability of establishing the session key (PESKY), to study the likelihood that a conference key can be established in the presence of budget constraints. Simulations show that the likelihood of forming a group key using a tree-based conference keying scheme is higher than the GDH schemes of Steiner *et al.*. Finally, we study the effect that greedy users have upon the Huffman-based conference keying scheme, and present a method to mitigate the detrimental effects of the greedy users upon the total cost.**

***Index Terms*—Conference key agreement, Diffie-Hellman, Huffman algorithm.**

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 group communication is protected from unwanted eavesdroppers. In order to protect the communication traffic, the information must be encrypted, requiring that the privileged parties share an encryption and decryption key. There are two basic approaches to establishing the group key: first, is to employ a key *distribution* protocol, where the formation of the key is performed by a single, centralized entity; or second, by employing a contributory key *agreement* protocol, where legitimate members exchange information that they use to agree upon a key.

In key distribution, the entity responsible for generating and distributing the group key is either an outside entity, such as a trusted third party (TTP), or an appropriately elected group member. The use of a single entity for key distribution is well-suited for applications where it is natural to have a single entity responsible for the group service. For example, one-to-many services involving large multicast groups, such as media services, are appropriate for centralized key distribution. The problem of centralized group key distribution has seen considerable attention recently in the literature [1]–[8]. A decentralized approach to key distribution was described in [8], where group participants generate group keys and distribute them amongst themselves.

The are many scenarios, however, where using key distribution is not appropriate. The use a TTP for distributing keys can be problematic as a TTP can act a single point of failure for the group's security. Further, in many cases it is not possible to have a third party administer the group key as there might not be a single entity that is trusted by all participants, or that has sufficient resources to maintain the intermediate key material for the entire group. An example might occur in ad hoc sensor networks where no single device has the storage resources needed to maintain intermediate keying material for the entire group. In such a case, *contributory* approaches are needed, where the group members each make independent contributions to the formation of the group key.

The classic example of a contributory scheme is the Diffie–Hellman (DH) key establishment scheme [9], in which two parties exchange messages that allow them to securely agree upon a key. Several researchers have studied the problem of establishing a *Diffie–Hellman-like* conference key [10]–[14]. 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 consisting of group members with different computational capabilities, pricing plans, and bandwidth resources. For these applications, minimizing the total bandwidth or amount of rounds might not be the most suitable metric. Instead, one should aim to minimize a cost function that incorporates the different costs or resource constraints of each user. The key generation scheme must therefore decide whether it is feasible to generate a key and if so determine a cost-aware procedure for generating the group key.

In this paper, we develop methods for establishing a Diffie–Hellman-like conference key for heterogeneous net-

Manuscript received January 14, 2002; revised May 9, 2003; approved by IEEE/ACM Transactions on Networking Editor S. Paul.

W. Trappe is with WINLAB, Rutgers, The State University of New Jersey, Piscataway, NJ 08852 USA (e-mail: trappe@winlab.rutgers.edu).

Y. Wang is with the Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688 USA (e-mail: yuke@ut-dallas.edu).

K. J. R. Liu is with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD USA 20742 (e-mail: kjrliu@eng.umd.edu).

Digital Object Identifier 10.1109/TNET.2004.842236

works. We start by reviewing the Diffie–Hellman protocol, and presenting several conference keying schemes that employ the Diffie–Hellman problem. In Section III, we present the butterfly scheme which builds the group key using the approach of [10], and show that an underlying tree, which we call the *conference tree*, governs the process by which subgroup keys are formed en route to establishing the group key. In Section IV, we consider the problem of designing a conference tree when the users have different capabilities. We first examine the case when the users have different costs. In this case, the optimal conference tree can be constructed using the Huffman algorithm. We then examine the problem of choosing a conference tree when the users have the same cost, but are subject to varying budget constraints. Next, we consider the more general case where the users have different costs as well as different budgets. A computationally efficient near-optimal algorithm is presented that determines a conference tree whose total cost is very close to the optimal performance achieved by conference trees determined using either full-search or integer programming techniques. In Section V, we present simulation results comparing the cost and likelihood of forming a group key using tree-based schemes and several existent schemes. From these simulations we conclude that the tree formulation for establishing a group key allows for great flexibility, and can efficiently establish group keys in resource-limited scenarios. Finally, in Section VI, we study the effects that the quantization and clipping of user costs have upon the total cost, and then investigate the effect that untrustworthy users can have upon the total cost of forming the group key using the Huffman-based conference tree. By choosing an appropriate clipping threshold level, the effects of miscoding are ameliorated. In Section VII, we summarize our results and present conclusions.

II. Group DH Overview

In the basic DH scheme, the operations typically take place in $\mathbf{Z}_p$ (the integers mod a prime p), or using the points on an elliptic curve [15]. For consistency of notation, we shall develop our results for the group $\mathbf{Z}_p$. An element g is chosen such that g generates a suitably large subgroup of $\mathbf{Z}_p$. Both party A and party B choose a private secret $\alpha_j \in \mathbf{Z}_p^*$ where $j \in \{A, B\}$ and $\mathbf{Z}_p^*$ denotes the nonzero elements of $\mathbf{Z}_p$. They each calculate $y_j = g^{\alpha_j}$ and exchange y_j with each other. Party A then calculates the key via $K = (g^{\alpha_B})^{\alpha_A} = g^{\alpha_B \alpha_A}$ and similarly for party B.

The problem of establishing a *Diffie–Hellman-like* conference key has been investigated by several others [10]–[12]. One of the first *Diffie–Hellman-like* conference key establishment schemes was proposed by Ingemarsson *et al.* [10]. In the Ingemarsson (ING) scheme, the group members are arranged in a logical ring (e.g. $A \rightarrow B \rightarrow C \rightarrow A$). In a given round, every participant receives a message from its left-hand neighbor, raises that to their exponent, and passes it to their right-hand neighbor. For example, in the first round of a three person group exchange, we have $A \rightarrow B : g^{\alpha_A}$, $B \rightarrow C : g^{\alpha_B}$ and $C \rightarrow A : g^{\alpha_C}$. Then, in the second round $A \rightarrow B : (g^{\alpha_C})^{\alpha_A}$, $B \rightarrow C : (g^{\alpha_A})^{\alpha_B}$, and $C \rightarrow A : (g^{\alpha_B})^{\alpha_C}$. Finally, the shared key is $g^{\alpha_A \alpha_B \alpha_C}$, which they each can calculate by raising the final received message to their private exponent. For n users this scheme requires $n - 1$ rounds.

Another notable scheme is the Burmester-Desmedt (BD) conference key scheme [11]. This scheme consists of three rounds. During the first round, each user u_j generates a random exponent α_j and broadcasts $z_j = g^{\alpha_j}$. The second round consists of each user u_j receiving z_j and broadcasts the quantity $x_j = (z_{j+1} z_{j-1}^{-1})^{\alpha_j}$. In the final round, each user u_j calculates the shared key $K = z_{j-1}^{n\alpha_j} x_j^{n-1} x_{j+1}^{n-2} \cdots x_{j-2}$. It can be shown that the shared key is actually the quantity $K = g^{\alpha_1\alpha_2 + \alpha_2\alpha_3 + \cdots \alpha_n\alpha_1}$. Although the BD scheme requires only three rounds to establish the group key, the actual communication efficiency is typically less. In general, it is impractical to support *simultaneous* transmission of messages by several users, or reception of several simultaneous messages by an individual user. In particular, for traditional networks employing a shared medium, such as Ethernet LAN's or 802.11 wireless networks, it is necessary to modify BD to use sequential broadcasts, which makes the amount of rounds needed linear in the amount of users.

In [12], the GDH.1, GDH.2 and GDH.3 protocols are described that extend the two-party DH scheme to the n-party case. The GDH.1/2 protocols consist of two stages: an upflow and a downflow stage. In the upflow stage of protocol GDH.1 user u_j receives a message of the form $\{g^{\alpha_1}, g^{\alpha_1\alpha_2}, \cdots, g^{\alpha_1 \cdots \alpha_{j-1}}\}$ and computes $g^{\alpha_1\alpha_2 \cdots \alpha_j}$ by taking the last element of the received message and raising it to the α_j power. User u_j then sends to user u_{j+1} the message $\{g^{\alpha_1}, g^{\alpha_1\alpha_2}, \cdots, g^{\alpha_1 \cdots \alpha_{j-1}}, g^{\alpha_1 \cdots \alpha_j}\}$. During the downflow stage, user u_n takes the output of the upflow stage, treats $g^{\alpha_1 \cdots \alpha_n}$ as the key, calculates g^{α_n} and raises the first $n - 2$ elements of the output of the upflow stage to the α_n power. Then user u_n sends user u_{n-1} a message of the form $\{g^{\alpha_n}, g^{\alpha_1\alpha_n}, \cdots, g^{\alpha_1 \cdots \alpha_{n-2}\alpha_n}\}$. User u_j performs likewise, calculating the key $g^{\alpha_1 \cdots \alpha_n}$ using the last term of the received message, and forward to u_{j-1} a message formed by taking the first $j - 1$ terms of the received message and raising them to the α_jth power. The GDH.3 scheme is a *centralized* scheme that differs from GDH.1/2 in that one user gathers contributions from all users, performs the majority of the computation for the group, and sends messages to each user that can be used to calculate the group secret. The *centralized* nature of the GDH.3 scheme is a drawback in environments where there is no single entity with significantly greater capabilities than the others users.

The amount of messages sent and received, as well as the amount of bandwidth consumed are important measures of a protocol's efficiency [12], [13]. Another important measure is the amount of rounds that a protocol takes to establish a group secret. A protocol that takes more rounds to establish a shared key is less favorable in environments where time is a precious resource and synchronization is difficult to maintain. In [13], the communication complexity involved in establishing a group key is studied. In this work, lower bounds for the total number of messages exchanged, as well as the amount of rounds needed to establish the group key, were determined.

The hypercube approach in [13] involves pairs of nodes performing the Diffie–Hellman protocol to establish a series of successive intermediate keys on the way to establishing the group key. A similar method has been proposed recently by several researchers to create a group Diffie–Hellman key using a tree structure [16]–[19]. For example, the TGDH scheme [16] establishes a group Diffie–Hellman key for dynamic peer groups by using a binary key tree. Every internal node on the key tree corresponds to the two children nodes performing Diffie–Hellman to establish an intermediate key. The amount of rounds required by TGDH is $\mathcal{O}(\log n)$, where n is the number of users. A different perspective was presented in [18], [19] in which 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 fast Fourier transform (FFT) calculations. Due to the relationship between the FFT and tree-based algorithms, the butterfly scheme may considered a tree-based group Diffie–Hellman scheme. However, unlike the methods of [16], [17], the butterfly scheme used the ING scheme as the basic building block, and provided a broader and more general family of approaches in which the amount of rounds needed to establish the group key is logarithmic in the group size.

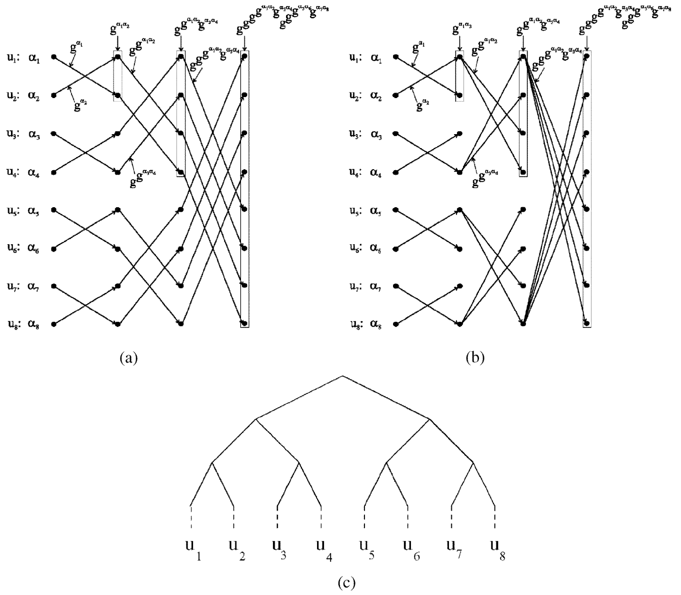


Fig. 1. The radix-2 butterfly scheme for establishing a group key for eight users. (a) Without broadcasts, (b) using broadcasts, and (c) the associated conference tree.

III. Conference Trees and the Butterfly Scheme

The general butterfly scheme is built using the ING scheme. However, since the two-party DH protocol is a special case of the ING scheme, we shall use the two-party DH protocol to introduce the basic ideas involved and then extend to using more general ING schemes. We refer to butterfly schemes built using two-party DH as radix-2 butterfly schemes. The term *radix* and *butterfly* is borrowed from the signal processing community, and their usage is motivated by the resemblance between the communication flow of our butterfly scheme, and the butterfly signal flow diagrams associated with FFT computations [20]. In our work, the usage of radix refers to the size of the initial subgroups used in the butterfly scheme.

We explain the basic idea behind the radix-2 butterfly scheme, by supposing that the number of users n is a power of 2. The users are paired up with each other to form two-person subgroups, and a key is established for each of these two-person subgroups using the conventional DH protocol. These subgroups are paired up with each other to form larger 4 member subgroups, and the two-party DH protocol is used to establish a group key for the 4 member subgroups. We successively group subgroups to form larger subgroups and use two-party DH to ultimately achieve a shared group key.

A formal description of the butterfly scheme for $n = 2^r$ members is as follows. Initially, suppose each user u_j has a random secret integer $\alpha_j \in \mathbf{Z}_p^*$. The n users are broken into pairs of users $u_j^1 = \{u_{2j-1}, u_{2j}\}$. Here we have used the superscript in the notation to denote which round of pairings we are dealing with, while the subscript references the pair. We also 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 $x_j^1 = g^{x_{2j-1}^0 x_{2j}^0} = g^{\alpha_{2j-1}\alpha_{2j}}$ (mod p). Since $x_j^1 \in \mathbf{Z}_p^*$, and both members of a pair have established a conventional DH key, we may now group 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 $u_j^2 = \{u_{2j-1}^1, u_{2j}^1\}$ so that 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 $x_j^2 = g^{x_{2j-1}^1 x_{2j}^1}$ (mod p). A third pairing, consisting of eight 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 $x_j^k = g^{x_{2j-1}^{k-1} x_{2j}^{k-1}}$ (mod p). The procedure continues until there are only two intermediate values that can be combined to get the group secret. We note that, although we shall refer to the final group secret as the group key, in practice this shared secret is actually used as input into a cryptographic one-way hash function to derive the actual group key.

A trellis diagram depicting the communication flows between users is depicted in Fig. 1(a). It is not necessary that each user communicate during each round. In fact, such an operation might use more power since many users are transmitting identical information. In networks, where multicasting is available, alternative trellis diagrams can be constructed where one user multicasts an intermediate message to multiple users. An example of such a trellis is depicted in Fig. 1(b). An alternative way to view the butterfly scheme is provided in Fig. 1(c), which depicts the tree associated with the butterfly scheme. This tree, which we refer to as the *conference tree*, describes the successive subgroups and subgroup keys that are formed en route to establishing the key for the entire group. For example, 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 can allow $\{u_1, u_2, u_3, u_4\}$ to communicate securely amongst themselves if so desired.

When n is not a power of 2, a group key still can be established easily. In this case, we form a subgroup with an amount of users equal to the largest power of 2 less than or equal to n.

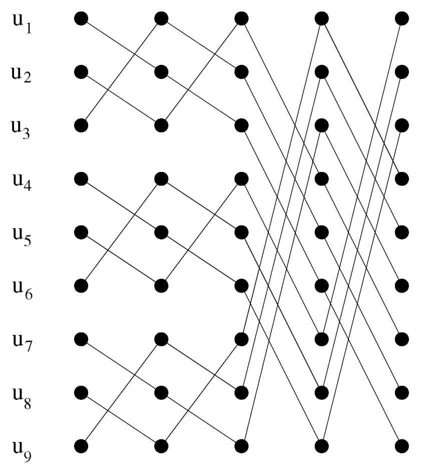


Fig. 2. The trellis for $n = 9$ users using two levels of three-party ING scheme.

We now extend the approach used above to employ the more general ING scheme as the basic building block. Since the resulting schemes are not built using a two-party protocol, they are termed nonradix-2 butterfly schemes. Suppose that $n = p_1 p_2 \cdots p_r$ is the number of users, and the p_j are not necessarily prime. The general ING butterfly scheme starts by breaking the group into subgroups of size p_1 and uses the ING scheme to establish a shared key for each of the $n_2 = p_2 \cdots p_r$ subgroups. The n_2 subgroups are further broken down into subgroups consisting of p_2 subgroups, and the ING protocol is used to establish subgroup keys for these larger subgroups. The process continues until a key is established for the entire group. An example is depicted for the case of $n = 9$ users in Fig. 2. The total amount of rounds is $2 \log_3 9 = 4$, and the amount of messages is 36. The direct use of the ING scheme for 9 users requires 8 rounds and 72 messages. The divide and conquer strategy in the butterfly approach improves the efficiency of the ING scheme. Additionally, the logarithmic amount of rounds needed by the butterfly scheme to establish the group key is an improvement over the linear amount of rounds required by the GDH schemes of [12]. We further note that the hypercube approach of [13] also requires a logarithmic amount of rounds to establish the conference key. However, the hypercube approach does not address the issue of using a general subgroup size as the building block for designing a scalable conference key establishment scheme. By using the ING scheme as the basic module in the butterfly scheme, we have generalized their approach. Further, the butterfly scheme allows for the use of multicast channels to improve communication efficiency.

It is not necessary to use a factorization of n in designing the nonradix-2 butterfly scheme. In fact, for prime n, this factorization would necessitate using an n-party ING scheme, and require a large amount of rounds in forming the group key. Rather, what is required is that the degrees p_j of the ING schemes used satisfy $\prod p_j \geq n$. In this case, some positions are left unused. For example, when $n = 8$ and $p_1 = p_2 = 3$ one position of a three-party ING scheme is empty, in which case that computation simply uses the 2-party DH scheme instead. If we require that all of the computations on one level of a conference tree are completed prior to the formation of the keys in the next level up the conference tree, then using the two-party DH scheme as the building block leads to trees with the least amount of rounds needed to establish the group key. The proof for this claim is provided in Appendix I. Since using two-party DH leads to binary trees that require the least amount of time rounds, we shall restrict our attention to binary trees for the remainder of the paper.

In the butterfly schemes described above, the conference trees were almost balanced and full. For example, the conference tree for $n = 8$ users involves three levels of internal nodes, and all eight users are placed at the same depth in the tree. For more arbitrary amounts of users, the users are all roughly placed at the same depth. In the next section, we shall exploit the extra freedom provided by more general binary conference trees by placing users at different depths to reduce the total group cost needed to form the group key.

IV. Conference Tree Optimization

It is important to study the problem of efficiently establishing a conference key while considering the varying user costs. To accomplish the efficient establishment of a conference key in a heterogeneous environment, we introduce a new entity, called the Conference Keying Assistant (CKA). The CKA is responsible for collecting the users' costs or budgets, determining the appropriate conference keying tree, and conveying the conference tree to the conference members if it is feasible to establish the group key. The CKA is not responsible for performing any service beyond the calculation and distribution of the appropriate conference tree, and therefore only needs to be a *semi-trusted* entity who will accurately convey the conference tree to the conference members. We note that the CKA may be a member of the conference.

In this section, we present methods that the CKA can employ to design the conference tree that used by the group members to establish 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. We present algorithms to efficiently determine the conference keys for each of these problems separately, and then together.

A. Minimizing Total Cost

First, assume that we have n users, and that each user u_j has a cost $w_j \geq 0$ associated with performing one two-party Diffie–Hellman protocol. There are several possible candidates for costs w_j:

- Energy Consumption: When the lifetime of the network is a concern, it is desirable to reduce the amount of energy used in forming the group key. There are two primary factors that contribute to the amount of energy used: computation and communication. The Diffie–Hellman protocol involves modular exponentiation, typically requiring several magnitudes more computation than cryptographic operations based upon symmetric ciphers, and hence requires considerable computational effort. Communication also plays an important role in many networks as both transmission and reception of messages requires significant energy [21].

- Storage Resources: Many networks might involve small devices with limited storage. It might therefore be natural to impose a cost associated with utilizing the storage resource as that storage might be important for other applications running on the device.
- Pricing: In many networks, each user might have different costs associated with using the network. For example, a group of users might have different pricing plans describing the monetary cost to transmit a message.

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 $C = \sum w_j l_j$ of this tree.

We first address the question of what is the minimum total cost necessary for establishing the group key for n users. This problem can be addressed using coding theory. If we define p_j as $p_j = w_j/(\sum_k w_k)$, then $\sum_j p_j l_j$ is just a scaling of $\sum_j w_j l_j$ by $W = \sum_k w_k$. Let us 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. We thus infer the following lower bound on the total cost for establishing a group key, which follows from the lower bound for the expected codelength of an instantaneous binary code for X [22]:

Lemma 1: Suppose that n users wish to establish a group secret and each user u_j has a cost w_j associated with performing one two-party Diffie–Hellman protocol. Then the total cost C of establishing the group secret satisfies $-W \sum_j p_j \log_2 p_j \leq C$ where $p_j = w_j/W$.

The observation that efficiently establishing a group key is related to coding allows the CKA to use procedures from coding theory to determine desirable conference trees. In particular, Huffman coding [23] is computationally efficient and yields the optimal conference tree that minimizes the total weighted cost. That is, if C^* is the cost of forming the group key using the Huffman tree with lengths l_j^*, 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 following bound $WH(p) \leq \sum_j w_j l_j^* < W(H(p)+1)$, 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.

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

Example 1: Consider a group of eight 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 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 with $l = (3,3,3,3,3,3,3,3)$, which corresponds to the Butterfly scheme, is 372.

We now quantify the improvement that is available when using the Huffman code compared to the cost of using an arbitrary conference tree. For an arbitrary conference tree, we suppose that the length assigned to user u_j is l_j. The expected length under the probability p_j of the code with lengths l_j satisfies [22]

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

where q is the probability distribution with $q_j = 2^{-l_j}$, and $D(p\|q)$ is the Kullback–Leibler divergence 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 bound of (1) with the bound for the cost of the optimal code $C^* < W(H(p)+1)$ to get $C - C^* > W(D(p\|q) - 1)$. When $D(p\|q) > 1$, this bound is an improvement over the trivial bound $C - C^* \geq 0$.

B. Budget Constraints

The parties wishing to establish a conference key might have a limited budget to spend. In these cases, rather than minimize the total cost, we should ensure that one can first establish the group key, and then reduce the total amount of resources 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 he is willing to participate in when establishing the group key. Without loss of generality, we assume that the users' budgets b_j satisfy $b_j \leq b_k$ for $j < k$. We define the budget vector as $b = (b_1, b_2, \cdots, b_n)$. The length vector $l = (l_1, l_2, \cdots, l_n)$ describes the lengths from each user's node to the root of the conference tree. 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 [22]:

Lemma 2: 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*. Using a feasible budget vector as the length vector does not always lead to a full conference tree in which every node has two children. To get a full tree, we must trim the budget vector to produce a length vector l that achieves the Kraft Equality. The length vector is formed by reducing elements of the budget 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 3: Suppose $b = (b_1, b_2, \cdots, b_n)$, with $b_j \leq b_k$ for $j < k$, satisfies the strict Kraft Inequality, $\sum_j 2^{-b_j} < 1$, then the modified budget vector c defined by $c = (b_1, b_2, \cdots, b_{n-1}, b_n - 1)$ satisfies the Kraft Inequality, $\sum_j 2^{-c_j} \leq 1$.

The proof of this lemma is provided in Appendix II. A consequence of this lemma is that if we subtract 1 from one of the b_j, then choosing the largest b_j least affects $\sum_j 2^{-b_j}$. Using this idea, Algorithm 1 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 $\sum_j 2^{-l_j} = 1$ and $\sum_j l_j$ is minimized over all length vectors

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

Algorithm 1. Algorithm for calculating the optimal length vector l.

c satisfying $\sum_j 2^{-c_j} \le 1$. The optimality of this algorithm is discussed in Appendix II.

Example 2: Consider a group of $n = 5$ users with a budget vector $b = (2,2,3,4,4)$. Then Algorithm 1 produces the following intermediate values for l on the way to calculating the final length vector $l = (2,2,2,3,3)$:

$$(2,2,3,4,4) \rightarrow (2,2,3,3,4), \rightarrow (2,2,3,3,3) \rightarrow (2,2,2,3,3).$$

C. Combined Budget and Cost Optimization

We now address the more realistic scenario where users have different costs as well as budget constraints. We are therefore interested in the problem of minimizing the total cost of the length assignments l_j for the weights w_j given the budget constraint $l_j \le b_j$. This problem is formally stated as:

$$\text{Minimize} \quad \sum_{j=1}^{n} w_j l_j$$
$$\text{subject to} \quad 1 \le l_j \le b_j, \quad \sum_{j=1}^{n} 2^{-l_j} = 1, \quad l_j \in \mathbf{Z}^+$$

where $\mathbf{Z}^+$ denotes the nonnegative integers. Once a length vector has been determined, it can be sorted in ascending order to describe a conference tree.

This problem is more difficult than either the minimum cost problem or the budget-constrained problem. If the budget vector is constant, i.e. $b_j = b$ for every j, then the methods of length-constrained Huffman codes may be applied. One efficient algorithm for finding the optimal code under the maximum codeword length constraint is presented in [24], which is based on the algorithm of [25]. A near optimal solution can be found using Lagrange relaxation, and an efficient implementation is described in [26]. However, in the more general case where the budgets vary from user to user, it is difficult to find the optimal solution since the ordering $w_j \le w_k$ does not imply $l_j \ge l_k$.

Two suboptimal approaches that employ a greedy strategy were developed to tackle the general problem where the budgets vary from user to user. The first algorithm, described in Algorithm 2, is a variant of Algorithm 1, which starts with a length assignment $l = b$ and chooses to decrease the element l_j of the length vector that most reduces the total cost $\sum w_k l_k$ at that step while maintaining the Kraft Inequality. This greedy algorithm is not optimal, as can be seen by the example $b = (2,2,3,3)$ with costs $w = (10,7,6,6)$. In this example, the algorithm produces the length vector $l = (1,2,3,3)$ (which has a total cost of 60), whereas the optimal length vector is $l^* = (2,2,2,2)$ (which has a total cost of 58).

```
Data    : A budget vector $b$.
if $\sum 2^{-b_j} > 1$ then
    No solution. ;
end
$l = b$ ;
while $\sum 2^{-l_j} < 1$ do
    Let $\delta = 1 - \sum 2^{-l_j}$ ;
    $K = -\lceil \log_2 \delta \rceil$ ;
    $J = \{j : l_j \ge K\}$ ;
    Let $j_0 = \arg\max_{j \in J} w_j$ ;
    $l_{j_0} = l_{j_0} - 1$ ;
end
```

Algorithm 2. Algorithm for calculating the length vector l, given budget b and costs w_j.

```
Data    : A budget vector $b$.
if $\sum 2^{-b_j} > 1$ then
    No solution. ;
end
$l = b$ ;
while $\sum 2^{-l_j} < 1$ do
    Let $\delta = 1 - \sum 2^{-l_j}$ ;
    $K = -\lceil \log_2 \delta \rceil$ ;
    $J = \{j : l_j \ge K\}$ ;
    Let $j_0 = \arg\max_{j \in J} w_j 2^{l_j}$ ;
    $l_{j_0} = l_{j_0} - 1$ ;
end
```

Algorithm 3. Improved algorithm for calculating the length vector l, given budget b and costs w_j.

Algorithm 2 is a naive greedy algorithm. By slightly altering this algorithm, another greedy algorithm may be developed with better performance. Instead of decreasing the element that best decreases the total cost, Algorithm 3 chooses to decrease the element with the largest value $w_j 2^{l_j}$. This corresponds to choosing the element that would have the largest change in the cost function per change in the Kraft Inequality. A similar strategy is often used in designing incremental resource allocation schemes in operations research [27]. Algorithm 3 is also suboptimal, but exhibits better performance than Algorithm 2 with a negligible increase in the amount of computation needed. The optimal solution to the combined budget and cost optimization problem can be obtained by performing either full-search, or using the methods of integer programming. One useful approach is to apply the branch and bound method to the problem [28].

To compare the performance of Algorithm 2 and Algorithm 3, we performed a simulation for n users, where each user's budget b_j was chosen uniformly from [1, $3n$], and weights w_j were chosen uniformly from [1,100]. We compared the performance between the length vectors found using Algorithm 2, which we denote $l^{(1)}$, and Algorithm 3, which we denote $l^{(2)}$, by looking at the relative cost difference $\rho = (C(l^{(1)}) - C(l^{(2)}))/C(l^{(1)})$. The quantity was calculated and averaged over 100 realizations. For a group size of $n = 50$ the relative improvement was 0.7202, for a group size of $n = 100$ the relative improvement was measured to be 0.8971, and for a group size of $n = 150$ the relative improvement was measured to be 0.9439. Similar simulations for different group sizes and different user budgets were performed, and Algorithm 3 consistently performed better than Algorithm 2.

A similar simulation was performed to compare the performance of Algorithm 3 with the optimal solution. Due to the computational complexity of finding the optimal solution, we compared the relative difference between the cost for using Algorithm 3 and the optimal solution for group sizes of $n =$ 5, 6, 7, 8, 9, 10, and 11. In all cases we saw that Algorithm 3 produces the group key with cost that is within 0.5% of the optimal cost.

Since determining the optimal solution is very computationally intensive for large group sizes, it is unreasonable for the CKA to find the optimal conference tree when users have both budget constraints and varying costs. Instead, Algorithm 3, although not optimal, has very competitive performance and its computational requirements are small compared to full-search or the branch and bound method, and is a reasonable candidate for the CKA to use in determining the conference tree.

D. Updating Keys Due to Membership Dynamics

Although this paper primarily focuses on the problem of efficient initial key agreement, there are many group scenarios where the group membership will change during the lifetime of the group communication. When a member joins or leaves the group, it is often necessary to change the keys in order to provide forward and backward integrity of keying material [2].

The problem of rekeying contributory key agreement schemes has been examined recently, [14], [16], [17], [29]. Both [16] and [17] addressed the issue of designing auxiliary protocols to handle the need to rekey a tree-based contributory key agreement scheme during member join and departure operations. The member join and departure protocols described in these two papers are essentially the same and can be applied with slight modification to rekey the conference key trees described in this paper.

For brevity, we will not repeat the member join and departure protocols here and instead refer the reader to the discussion of the rekeying protocols in [16], [17]. During a member departure operation, there is no control over *who* is leaving the group, and hence no control over *where* that departing user is located on the tree at that time. During member departure, all of the intermediate keys on the path from the departing user to the root of the tree must be changed. Since the neighbors to this path, as well as their costs, are already fixed, the total cost needed to rekey those adjacent intermediate keys is also fixed. Hence, the member departure operations described in [16], [17] do not require modification to rekey the conference trees described in this paper.

Although it is not possible to control the position of the departing member, we do have control over where we place a joining user in the tree. Consider a group of n users with costs w_j who have already established a group key according to a conference tree. In order for the group to establish a new group key using the least amount of cost possible, we simply add the new user to the top of the existing conference tree and form a new group key by performing one round of Diffie–Hellman. That is, the rekeying protocol would have the new user u_{n+1} perform a single Diffie–Hellman key exchange with users $u_1, u_2, \cdots, u_n$. Since each user incurs their cost w_j for participating in the Diffie–Hellman key establishment, the total cost incurred is $\sum_{j=1}^{n+1} w_j$. It is easy to see that placing the new user anywhere else on the tree would lead to a larger member join cost.

Thus, it is a simple task to minimize the cost for adding a new member to the group. However, by increasing the height of the tree by one for every user, we have now increased the cost needed for any future member departure. On the otherhand, if we had added a member to any terminal node of the tree and then performed a member departure operation according to [16], [17], we would have had a higher member join cost, but the member departure cost would have been less since there would have been users whose cost to handle a member departure would have been the same as if no user had joined.

To handle these conflicting issues, it is possible to search for a position on the existing tree for the new user to join that jointly considers the cost of joining and the additional leave cost the new user would impose on existing users. Suppose we label the terminal nodes of the existing conference tree as t_x, and that users are equally likely to depart. One possible method for jointly addressing join and additional user departure costs is to place the joining user at a position that minimizes the joint cost function $C_T(t_x) = \rho C_{MJ}(t_x) + (1-\rho)\Delta C_{ML}(t_x)$, where $C_{MJ}(t_x)$ is the cost incurred for the new user u_{n+1} to join at node t_x, and $\Delta C_{ML}(t_x)$ is the average additional leave cost incurred by user u_{n+1} joining at node t_x. We take $\rho \in [0,1]$ to be a factor that weights the importance of the cost of member join compared to the cost of potential member leaves. Minimizing C_T involves searching the n terminal nodes t_x of the existing conference tree. Rekeying then involves splitting node t_x and following the member join procedure described in [16], [17].

V. Efficiency and Feasibility Evaluation

We compare our tree-based conference key establishment schemes with other schemes in the literature. We assume that no broadcast channels are available, and that if one user desires to communicate amongst many, he must establish many separate connections. There are two evaluations that we present: first, we consider the total cost needed to establish a group key when the users have different costs; second, we examine the feasibility of establishing a conference key when group members have different budget constraints.

A. Comparison of Total Cost

We simulated a scenario in which there were three classes of users: powerful users who have a low user cost, medium-powered users with moderate user costs, and low-powered users with a high user costs. In order to represent this distinction, the users were assumed to have weights drawn according to three different distributions. For every 10 users, 2 users have weights drawn according to the first distribution, 5 according to the second distribution, and 3 according to the third distribution. The first weight distribution was a discrete uniform distribution

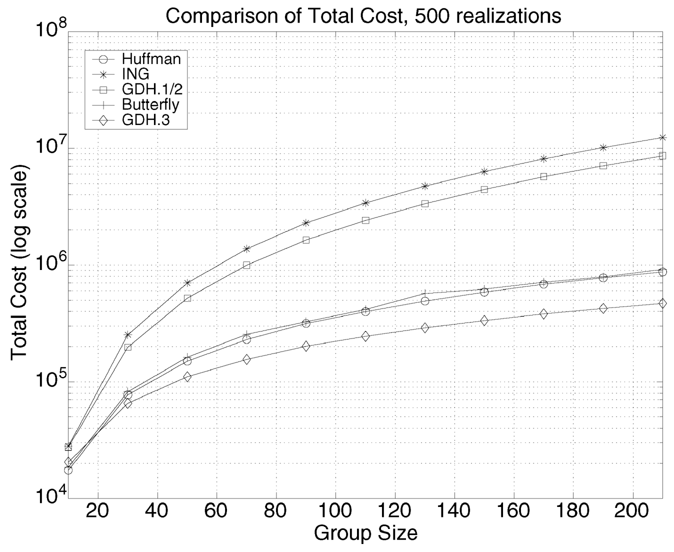


Fig. 3. Cost comparison of establishing a conference key using the Huffman-based conference tree, the ING scheme, GDH.1/2, the butterfly scheme, and the GDH.3 scheme.

with integer values from [1,50], while the second was a discrete uniform distribution over [501,550], and the third was a discrete uniform distribution over [951,1000].

We compared the total cost for the Huffman scheme with the cost of the butterfly scheme, the ING scheme, the GDH.1/2 scheme, and the GDH.3 scheme. We have chosen to focus our simulations on reducing the usage of computational resources associated with forming the group key. Similar simulations can be done to focus on reducing the usage of communication resources by simply considering the amount or size of messages exchanged in each scheme.

Since there are differences between the communication and computational procedures of the different schemes, we assume that the user costs are associated with the cost to perform the two modular exponentiations needed in a two-party DH scheme. This means, for example, that if a user has a cost of w to perform one round of two-party DH, then he has a cost of $3w/2$ to perform a three-party ING scheme since there are three modular exponentiations involved. We also assume that every user in a DH scheme performs the two modular exponentiations. For example, if the subgroup $\{u_1, u_2\}$ share a secret x and the subgroup $\{u_3, u_4\}$ share a secret y, and use DH to establish a shared key for the 4 members, then both u_1 and u_2 calculate g^x and g^{yx}. Similarly, both u_3 and u_4 calculate g^y and g^{xy}. In actuality, however, only one member from each subgroup must calculate and transmit the message g^x or g^y. The costs for the Huffman and butterfly schemes that we report do not reflect this possible savings, and are therefore overestimates of the actual costs.

The total cost required to establish the conference key was calculated for different group sizes and averaged over 500 realizations. The average costs are depicted in Fig. 3. Examining Fig. 3 we see that the ING and GDH.1/2 schemes have higher total cost than the Huffman, butterfly, and GDH.3 schemes. In this example, the Huffman scheme performs better than the butterfly scheme by an average of 6.7%. GDH.3 has the best performance in terms of total cost. However, GDH.3 is a *centralized* scheme and cannot be categorized as a completely *distributed* conference keying scheme since one user performs the majority of the computations for the group. In contrast, the Huffman scheme and the butterfly scheme are contributory and do not make any single user responsible for the majority of the computation (although they do allot more load to some users than others). In scenarios where it is appropriate to have one user or entity do nearly all of the work for the remaining users the use of centralized multicast key distribution schemes [2], [3], [6] will lead to more efficient distribution of keying information than conference keying schemes.

B. Feasibility Comparison

When the users have different budgets, it might not be possible for different schemes to establish a conference key. We shall quantify the likelihood that a conference key can be established in a scenario where the users' budgets are drawn according to a distribution by introducing the PESKY (Probability of Establishing the Session KeY) measure.

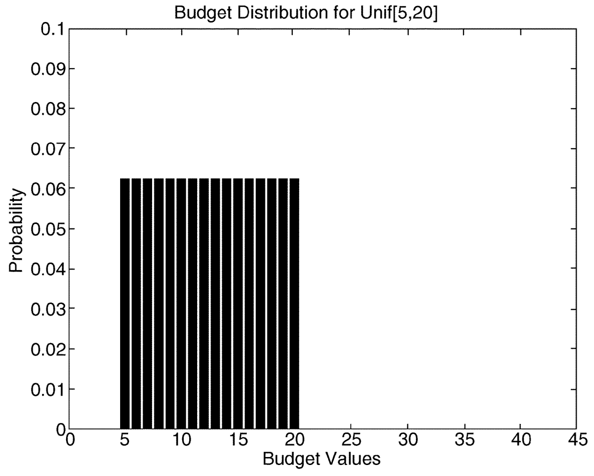
Suppose that $\mathcal{B}$ denotes the set of all possible budget vectors for n users, and that μ is a probability distribution over $\mathcal{B}$ describing the likelihood of the users having a certain budget vector. Let a conference key scheme be denoted by F, and $F(\mathcal{B})$ the set of all budget vectors $\mathcal{B}$ which are feasible for F. Then formally, the PESKY measure is defined as:

$$\text{PESKY}(F, n) = \sum_{b \in F(\mathcal{B})} \mu(b). \tag{2}$$

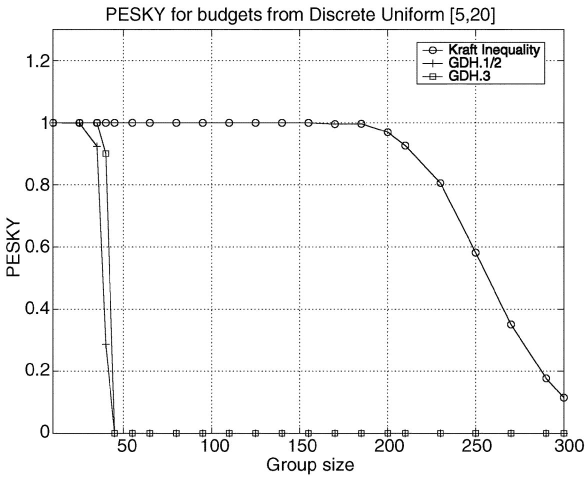
For example, if we let F refer to a conference tree scheme built using Algorithm 1, Algorithm 2, or Algorithm 3, then a budget vector is feasible if it satisfies the Kraft Inequality, and therefore $F(B) = \{b : \sum_j 2^{-b_j} \le 1\}$. In general, it is difficult to find closed form expressions for PESKY, and simulations may be used to estimate of PESKY.

We used PESKY to study the likelihood that different schemes could produce a group key when the user's budgets were drawn according to different distributions. We assumed that the budgets b_j correspond to the amount of two-party DH schemes that a user is willing to participate in, and that the two modular exponentiations are the most significant expense for the user. Therefore, each value of the budget allows for 2 modular exponentiations to be performed. We do not assume broadcasting, and instead assume that every user in a subgroup performs both of the modular exponentiations in a DH scheme. We compared the PESKY for Algorithms 1–3, with the PESKY for both the GDH.1/2 and GDH.3 schemes for several different budget distributions, and present two representative distributions. Since the PESKY for Algorithm 1, Algorithm 2, and Algorithm 3 are identical and are determined by the likelihood that a budget vector satisfies the Kraft Inequality, we will use the Kraft Inequality label in our figures to collectively refer to their PESKY.

The first budget distribution is a discrete uniform distribution with integer values from [5,20]. The distribution is presented in Fig. 4(a), and the corresponding PESKY curves are presented in Fig. 4(b). Since the GDH schemes require that one user performs an amount of modular exponentiations equal to the amount of users n, it is impossible for groups of more than

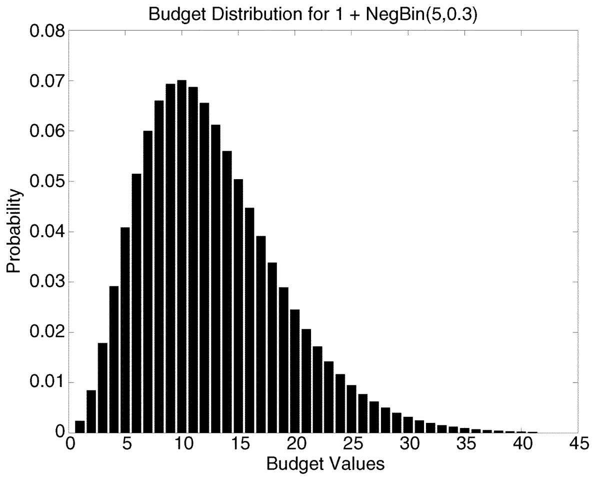


(a)

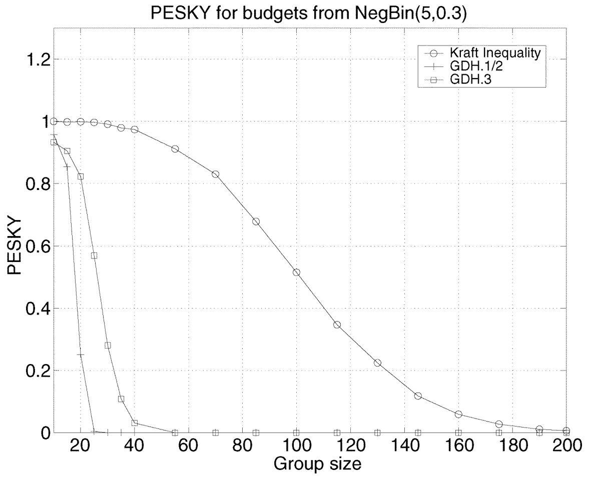


(b)

Fig. 4. (a) Budget distribution discrete uniform with integer values from [5,20]. (b) Corresponding PESKY.



(a)



(b)

Fig. 5. (a) Budget distribution, shifted version of a negative binomial distribution with parameters $s = 5$, and $p = 0.3$. (b) Corresponding PESKY.

40 users to be formed via the GDH protocols with this distribution, as can be seen in Fig. 4(b). The PESKY plots for this distribution demonstrate that it is more likely that a budget vector can satisfy the Kraft Inequality than the requirements of either the GDH.1/2 or GDH.3 schemes. It is not until the group sizes become larger than $n = 200$ that a significant decrease is observed in the likelihood of forming a group key using a conference tree.

In the second distribution, the budgets were drawn according to $1 + NegBin(5, 0.3)$, where $NegBin(s, p)$ is the negative binomial distribution with probability mass function $q(b) = \binom{s+b-1}{b} p^s (1-p)^b$ for $b \in \{0, 1, \cdots\}$. The addition of 1 was to ensure that no users had a budget of 0. The negative binomial distribution was chosen for the budget values since it is integer-valued, and its shape is easily tunable by adjusting the s and b parameters [30]. The distribution is depicted in Fig. 5(a), and the corresponding PESKY measures are depicted in Fig. 5(b). This distribution describes a similar phenomenon to the uniform distribution above, but includes a heavier tail at higher budget values that could represent a diminishing class of more powerful users. The fact that roughly 6% of this distribution corresponds to budget values below 5 has a significant effect upon the PESKY plots. When there are $n = 70$ users there is only an 80% chance of forming a conference key using one of these schemes with this distribution compared to a 100% chance with the distribution of Fig. 4(a). We also see that the GDH.1/2 schemes are very unlikely to successfully establish a group key, even for group sizes of $n = 20$, and that all of the GDH schemes are unable to establish a group key for groups of more then 60 users.

Therefore, in resource-limited scenarios, the choice of which conference keying scheme is very critical. The GDH.3 scheme, although cost-efficient, obtains this efficiency at the expense of requiring a single user have significantly more power and resources than the other users. In applications where the users have a more balanced distribution of resources, the GDH schemes have PESKY graphs that rapidly drop off and are therefore unlikely to successfully establish a group key. In these cases, the estimates of PESKY for tree-based conference keying schemes indicate that they are more likely to establish a group key, and Algorithm 3 is a judicious choice for constructing the conference tree since it requires little computational effort and has near-optimal performance.

VI. System Sensitivity to False Costs

In this section, we examine the effect that announcing costs different from the true user costs has upon the total cost of using the Huffman conference tree. First, we consider the issue that users announce costs that are approximations of the true costs. Next, we examine the case where some of the users are untrusted, and announce large costs for the purpose of reducing their individual cost. We present an approach that controls the detrimental effect that greedy users have upon the total cost.

A. Sensitivity to Approximate Costs

We begin by considering that the true user costs are $\hat{w}_j \in [1, \hat{B}]$, where $\hat{B}$ is a suitable upper bound placed on the exact costs. We suppose the costs that the users announce are derived by applying an operator T to $\hat{w}_j$, i.e. $w_j = T(\hat{w}_j)$. We define $\hat{W} = \sum_j \hat{w}_j$, and $\hat{p}_j = \hat{w}_j/\hat{W}$. If we build a code using p_j with lengths l_j, then the average length under $\hat{p}$ is $\sum_j \hat{p}_j l_j$. We show that if we design the code to minimize $\sum p_j l_j$, then we can design the operator T such that $\sum |\hat{p}_j l_j - p_j l_j|$ is small. Since $l_j \leq n$, we get $\sum_{j=1}^{n} |\hat{p}_j l_j - p_j l_j| \leq n(\sum_{j=1}^{n} |\hat{p}_j - p_j|)$. We now derive a bound for $\sum_{j=1}^{n} |\hat{p}_j - p_j|$:

$$\sum_{j=1}^{n} |\hat{p}_j - p_j| = \sum_{j=1}^{n} \left| \frac{w_j}{W} - \frac{\hat{w}_j}{\hat{W}} \right| \tag{3}$$

$$\leq \frac{1}{W\hat{W}} \left[\sum_{j=1}^{n} \left| w_j(\hat{W} - W) \right| + \sum_{j=1}^{n} |W(w_j - \hat{w}_j)| \right] \tag{4}$$

$$\leq \frac{1}{W\hat{W}} \left[2W \sum_{j=1}^{n} |w_j - \hat{w}_j| \right] = \frac{2}{\hat{W}} \left[\sum_{j=1}^{n} |w_j - \hat{w}_j| \right]. \tag{5}$$

In this derivation, we have made use of the fact that $|\hat{W} - W| = |\sum_j (\hat{w}_j - w_j)| \leq \sum_j |\hat{w}_j - w_j|$. We consider two cases for the operator T. The first case we consider is when T is a clipping operator, namely

$$T_{\hat{B}}(\hat{w}) = \begin{cases} \hat{w}: & \hat{w} \leq \hat{B} \\ \hat{B}: & \hat{w} > \hat{B} \end{cases}.$$

It is clear that as $\hat{B} \to \infty$, we have more $w_j = T_{\hat{B}}(\hat{w}_j) = \hat{w}_j$, and thus the bound (5) tends to 0 as we increase $\hat{B}$. We shall examine the clipping operator later in this section. The second operation we consider is quantization. Here we consider the interval $[1, \hat{B}]$ divided into N equally sized quantization bins. The operator T then maps $\hat{w}$ to the nearest quantization value, and $|w_j - \hat{w}_j| \leq \hat{B}/(2N)$. In this case, we get $\sum_{j=1}^{n} |\hat{p}_j - p_j| \leq (1/\hat{W})(\hat{B}n/N)$ which tends to 0 as the number of quantization bins N increases. Therefore, in both the case of clipping and quantization, the parameters can be adjusted to bring the probability distribution p close to $\hat{p}$, and thus the designed average codelength $\sum p_j l_j$ close to the average codelength of using l_j under $\hat{p}$.

B. Sensitivity to Costs From Untrustworthy Users

In many scenarios, there may be a user that hurts other users by either selfishly making his cost small, or maliciously trying to make the total cost of the remaining users large. 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 $l_1^* \leq l_2^* \leq \cdots \leq l_n^*$ [22]. Therefore, if a user would like to keep his cost as small as possible, he should announce as large of a weight as possible. Additionally, announcing a large weight causes the p_j of the other users to decrease, thereby increasing their codelengths (see [31] for the relationship between a symbol's codelength and its self-information). Thus, if a malicious user wishes to adversely affect the lengths of the other users, he should announce as large of a weight as possible.

We first derive the worst-case effect that one user can have upon the costs of the other group members when Huffman coding is used to construct the conference tree. We suppose that the untrustworthy user is u_1, and that he publishes a large weight w_1. To determine how much extra cost a large w_1 imposes upon the other $n-1$ users, we define $\check{W} = \sum_{k=2}^{n} w_k$ and define the probability $q_j = w_j/\check{W}$ 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 $\check{l}_j^*$ be the optimal codelengths constructed using q_j. Since u_1 is not involved in the construction of $\check{l}_j^*$, we have $\check{l}_1^* = 0$.

We define the following quantities: $C^* = \sum_{j=1}^{n} w_j l_j^*$, $\check{C}^* = \sum_{j=2}^{n} w_j \check{l}_j^*$, 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 $\check{C}^*$, which is the total cost of the $n-1$ users $u_2, u_3, \cdots, u_n$ without considering u_1's announced cost.

Since $\check{C}^*$ arises as the optimal code for the $n-1$ users with costs $w_2, w_3, \cdots, w_n$, we know $\check{C}^*$ minimizes costs of the form $\sum_{j=2}^{n} w_j l_j$. In particular, C_{ex}^* must satisfy: $C_{ex}^* = \sum_{j=2}^{n} w_j l_j^* \geq \sum_{j=2}^{n} w_j \check{l}_j^* = \check{C}^*$. We may derive an upper bound for C_{ex}^* by observing that the code given by $\check{l}_j^*$ can be used to construct a code for p_j by taking $l_1 = 1$ and $l_j = \check{l}_j^* + 1$. The optimal code for the weights $w_1, w_2, \cdots, w_n$ must be better than this code, and hence $C^* \leq w_1 + \sum_{j=2}^{n} w_j(\check{l}_j^* + 1) = \check{C}^* + W$. Since $C_{ex}^* = C^* - w_1 l_1^*$, we have $C_{ex}^* \leq \check{C}^* + W - w_1 l_1^* \leq \check{C}^* + \check{W}$. Gathering the results together, we get the overall bound $\check{C}^* \leq C_{ex}^* \leq \check{C}^* + \check{W}$. The upper bound is achieved when $w_1 > \check{W}$, and hence, in the worst case, u_1 forces the other $n-1$ users to spend an extra $\check{W}$ of resources.

Next, we consider the more general case where a fraction of the users are untrustworthy and announce large costs. Suppose that the true costs are $\hat{w}_j$, and that the announced costs are $\tilde{w}_j$. If the underlying statistics governing $\hat{w}_j$ are known, it is possible to determine which $\tilde{w}_j$ are outliers and remove those users from the group key formation procedure. However, in many cases, the value of the conference exists regardless of whether a few users were untrustworthy, and it is desirable to have those users in the conference. In this case, an approach must be used to reduce the detrimental effect of these bad users upon the cost of forming the entire group key.

We suppose that the CKA applies a clipping operator to the announced user costs $\tilde{w}_j$ to produce costs $w_j = T_B(\tilde{w}_j)$ that are used by the CKA in determining the conference tree. Ideally, we would like to build the conference tree using the exact costs $\hat{w}_j$, but these are not available. Instead, if the conference tree is built using w_j or $\tilde{w}_j$, the corresponding lengths l_j and $\tilde{l}_j$ are

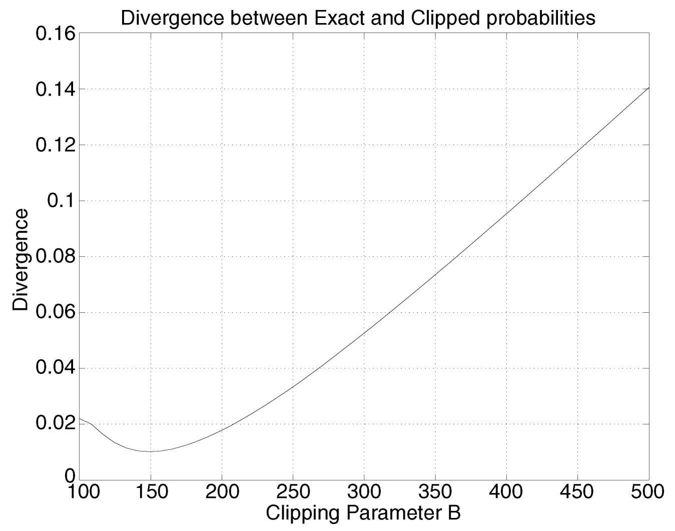


Fig. 6. An example divergence $D(\hat{p}||p)$ where $\hat{w}_j \sim 10LN(0,1)+100$, and $w_j = T_B(\hat{w}_j)$.

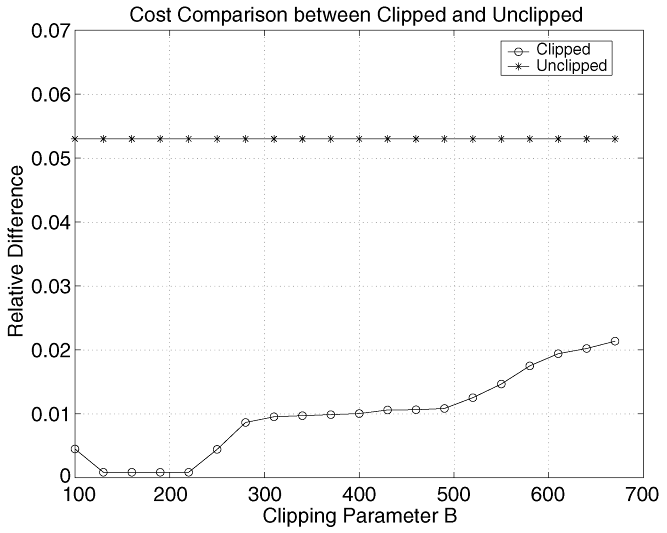


Fig. 7. The relative costs are presented for when the exact user costs are drawn as $\hat{w}_j \sim 10LN(0,1) + 100$. There is a 0.05 likelihood that a user is untrustworthy, and $Y = 1000$.

used with the exact costs $\hat{w}_j$, which can lead to an increase in the total cost.

To study the amount of additional cost incurred by using a code designed for w_j when the true costs are $\hat{w}_j$, we shall examine the average codelength. Hence we design codes for $p_j = w_j/W$ and $\tilde{p}_j = \tilde{w}_j/\tilde{W}$, where $\tilde{W} = \sum \tilde{w}_j$. We are interested in studying $\sum \hat{p}_j l_j$ and $\sum \hat{p}_j \tilde{l}_j$. The Kullback–Leibler divergence $D(\hat{p}||p)$ describes the additional average codelength that different coding schemes incur when designed for the wrong distribution p when the correct distribution is $\hat{p}$ [22], [31]–[33]. Given a model distribution for the true user costs, the CKA can use $D(\hat{p}||p)$ to determine the value of the clipping parameter B that minimizes the miscoding penalty.

We calculated the divergence for $n = 100$ users when the original costs w_j were drawn according to $10LN(0,1) + 100$, where $LN(\mu, \sigma)$ is the lognormal distribution arising from a normal distribution with mean μ and variance σ. The lognormal distribution was chosen because it has a long tail. The probability that a user is untrustworthy was 0.05, and untrustworthy users were assumed to announce a cost $\tilde{w}_j = \hat{w}_j + Y$, where $Y = 1000$ and $w_j = T_B(\tilde{w}_j)$. The choice of $Y = 1000$ was arbitrary and chosen to represent a large bias that an untrustworthy user might place on his announced costs. An example divergence $D(\hat{p}||p)$ for costs $\hat{w}_j$ drawn according to this distribution is presented in Fig. 6. There is a minimum that appears at approximately $B^* = 150$. A system should be designed for the average case. For costs drawn according to $10LN(0,1) + 100$, we averaged the optimal clipping value over 10 000 realizations and found the mean optimal clipping value to be $\overline{B}^* = 150.44$ and the variance of the optimal clipping value as $\sigma_{B^*} = 25.60$.

The relative difference between the cost of using the Huffman-based conference tree using w_j and $\hat{w}_j$ are now compared. If $\hat{l}_j$ are the optimal codelengths using $\hat{w}_j$, $\tilde{l}_j$ are the optimal codelengths constructed using $\tilde{w}_j$, and l_j are the optimal codelengths constructed using w_j, then we are interested in comparing $\rho = (\sum_j \hat{w}_j l_j - \hat{w}_j \hat{l}_j)/(\sum_j \hat{w}_j \hat{l}_j)$ and $\tilde{\rho} = (\sum_j \hat{w}_j l_j - \hat{w}_j \tilde{l}_j)/(\sum_j \hat{w}_j \hat{l}_j)$. We calculated these values for the case when the exact costs were drawn according to $10LN(0,1) + 100$ with $Y = 1000$, while the probability of a user being untrustworthy was 0.05. The results were averaged over 100 realizations and are presented in Fig. 7. The quantity ρ is presented for different clipping parameter values, and we observe that there is a range of minimal values from $B = 140$ to $B = 220$, which is roughly the region that the divergence curves predict. The clipped relative costs show a significant improvement over the unclipped relative costs. Without clipping, the untrustworthy users force the entire group to spend an average of over 5% more than if the exact user costs were used. By performing the clipping operation, however, this detrimental effect can be significantly lessened to less than 0.5%.

VII. Conclusion

In this paper we presented methods for establishing a conference key that are based upon the design of an underlying tree called the conference tree. In heterogeneous environments, where users have varying costs and budgets, the conference tree can be designed to address the user differences. We studied the problem of minimizing the total cost of establishing the group key when the users had different costs. The problem of designing the conference tree was related to source coding, and techniques for designing source codes, such as Huffman coding, were employed to design the conference tree. The second case we investigated was when the users had the same cost, but different budget requirements. A necessary condition for a conference tree to exist for a given vector of budget requirements is that the budget vector satisfies the Kraft Inequality. Finally, the third case we examined is when the users have both varying costs and budget requirements. We presented a computationally efficient near-optimal algorithm using a greedy incremental resource assignment strategy that achieves a total cost within 0.5% of the optimal solution for small group sizes.

In situations where no single user has an extremely large budget, centralized conference keying schemes are unlikely to successfully establish a conference key. To investigate this phenomenon, we introduced the PESKY measure, which describes the probability that a conference keying scheme can establish a session key in the presence of budget constraints. We provided simulations where the user budgets were drawn according to different distributions, and in all cases the PESKY values for different group sizes were higher for our tree-based schemes than for either the GDH.1/2 or the GDH.3 schemes.

Next, we examined the effect that using false user costs would have on the total cost. It was shown that by increasing the quantization resolution, or by increasing the threshold level, that the difference between the total cost of using the exact and approximate costs for a given length assignment tends to 0. We then examined the effect a subset of users who falsely announce large costs has upon the total cost. In order to reduce the detrimental effect of designing a conference tree for falsely announced user costs, we proposed the use of a clipping operator to prevent untrustworthy users from being too greedy and minimize the divergence to determine the optimal threshold value.

Appendix I

The total number of rounds needed in the ING butterfly scheme for $n = p_1 p_2 \cdots p_r$ users is $TR = \sum_{j=1}^{r}(p_j - 1) = (\sum_{j=1}^{r} p_j) - r$. When choosing a factorization for n, more factored representation lead to fewer rounds. We now show that using a binary conference tree produces the group key in the fewest amount of rounds. To do this, we show that if one uses a p_j ING scheme for round j of the group key establishment, then the use of several two-party DH schemes in place of the p_j ING scheme either produces the same amount of rounds or fewer in establishing the group key.

Lemma 4: Let n be the amount of users, and suppose that we wish to establish a conference tree where level j uses a p_j ING scheme as the basis, then a binary tree (where $p_j = 2$) produces an optimal conference tree.

Proof: Suppose that you have an optimal set of numbers $\{p_1, \cdots, p_r\}$ that are used to construct the conference tree for n users. Then the number $N = \prod_{j=1}^{r} p_j \geq n$, and the total rounds $TR = (\sum_{j=1}^{r} p_j) - r$ is minimal.

We will show that if there is a $p_j \neq 2$ then we may replace p_j by a sequence of numbers all of which have value 2. Suppose there is a j such that $p_j \neq 2$, then the p_j contributes $\Delta_j = p_j - 1$ to the total amount of rounds TR. Define $p'_j = \{2, 2, \cdots, 2\}$ which is a sequence of length $\lceil \log_2 p_j \rceil$. If we use this set of numbers in place of p_j, we instead contribute $\Delta'_j = \lceil \log_2 p_j \rceil$ to the total cost. It is clear that using p'_j in place of p_j produces an $N' \geq n$. However, the incremental cost $\Delta'_j = \lceil \log_2 p_j \rceil$ is less than or equal to Δ_j (in fact, if $p_j = 3$ then equality holds, else it is strictly less). Thus, if $p_j > 3$ then replacing p_j by p'_j produces a set of numbers with lesser amount of total rounds TR, which contradicts optimality. On the otherhand, if $p_j = 3$ then replacing p_j by p'_j will produce a set of numbers with an equal amount of total rounds TR, and hence we may choose to use p'_j instead of p_j in the construction of the optimal tree. By applying this argument to all $p_j \neq 2$ we conclude that a binary tree must produce an optimal tree. ■

The argument used above does not produce the optimal tree, but rather only implies that the optimal tree is binary. For example, consider $n = 27$. The total amount of rounds using three levels of three-party ING is $TR = 6$. If we use the above technique, we replace each 3 by $2 \cdot 2$, and get a conference tree with 2^6 terminal nodes and total cost of 6. However, the optimal tree in this case is the binary tree of depth $\lceil \log_2 n \rceil$, with total rounds $TR = 5$.

Appendix II

Lemma 3: Suppose $b = (b_1, b_2, \cdots, b_n)$ with $b_j \leq b_k$ for $j < k$ satisfies the strict 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 $\sum 2^{-c_j} \leq 1$.

Proof: Observe that 2^{b_n} is the common denominator of the set 2^{-b_j}. Thus $\sum 2^{-b_j}$ can be expressed as $\sum 2^{-b_j} = (x_1 + x_2 + \cdots + x_n)/2^{b_n} < 1$ where $x_j = 2^{b_n - b_j}$. In particular, $x_1 + x_2 + \cdots + x_n < 2^{b_n}$, and as a consequence $x_1 + x_2 + \cdots + (x_n + 1) \leq 2^{b_n}$. However, $(x_n + 1)/2^{b_n} = 1/2^{b_n - 1}$, and so the sequence $(b_1, b_2, \cdots, b_n - 1)$ satisfies the Kraft Inequality. ■

Lemma 4: Algorithm 1 produces an optimal length assignment vector l to the problem

$$\left\{ \min_{l} \sum_{j} l_j \ : \ 1 \leq l_j \leq b_j, \sum_{j} 2^{-l_j} \leq 1, l_j \in \mathbf{Z}^+ \right\}. \quad (6)$$

Proof: We will show that there is an optimal solution in which one decreases the largest value of the budget vector by one. Let l^* be an optimal solution to the problem. Then by the previous lemma $\sum 2^{-l_j^*} = 1$. Consider a sequence of steps that take the budget vector b to the optimal length vector l^* by decreasing one element by 1 during each step. We denote by J_* the sequence of indices involved in going from b to l^*, where $J_*(k)$ refers to the index of the budget vector that is decreased during kth step. Let j_0 be the index of the largest element of b, we claim there is an optimal solution l' with a corresponding $J_*(1) = j_0$. If $J_*(1) = j_0$ then we are done. However, if $J_*(1) \neq j_0$ then there are two cases. The first case is that there is another element of J_* that has value j_0, in which case we may switch that element with $J_*(1)$ to produce a new sequence of steps that does not alter the value of $\sum 2^{-l_j^*}$ and maintains the optimality of $\sum_j l_j$. The second case is that $j_0 \notin J_*$. If there are any other elements of b with the same value as b_{j_0}, then indices of these may be used in place of j_0, and considered in the preceding argument. However, if there are no b_j's with the same value as b_{j_0} then we seek a contradiction as to the optimality of l^*. Choose an arbitrary element of J_*. This element, which we denote by $J_*(k)$, by assumption has the property that it $b_{J_*(k)} < b_{j_0}$. Define $J_*^- = \{J_*(1), \cdots, J_*(k-1), J_*(k+1), \cdots\}$, which corresponds to the sequence of steps involved in J_* excluding the kth step. Define $J_\sharp = j_0 || J_{*-}$, which describes a new sequence of steps that starts with j_0 and then the steps of J_{*-}. Then $J_\sharp$ leads to a length vector $l^\sharp = l^* + e_{J_*(k)} - e_{j_0}$, where e_j is the vector of all zeros except in the jth index which has value 1. This length vector has the property that $\sum 2^{-l_j^\sharp} < \sum 2^{-l_j^*}$ since $2^{-l_{j_0}^*} < 2^{-l_{J_*(k)}^*}$. Hence $\sum 2^{-l_j^\sharp} < 1$. However, by the preceding lemma, this means that $l^\sharp$ can be used to produce a better length vector, which contradicts the optimality of l^*.

Hence, the optimal solution may as well have the first step reduce the largest element of the budget vector. Now the problem reduces to finding an optimal solution to the new budget $b' =$

$b - e_{j_0}$. By induction on the number of steps, we therefore conclude that choosing the largest element during each step yields an optimal solution, and hence the greediness of Algorithm 1 is optimal. ■

References

[1] M. J. Moyer, J. R. Rao, and P. Rohatgi, "A survey of security issues in multicast communications," *IEEE Network*, vol. 13, no. 6, pp. 12–23, Nov.–Dec. 1999.

[2] R. Canetti, J. Garay, G. Itkis, D. Miccianancio, M. Naor, and B. Pinkas, "Multicast security: a taxonomy and some efficient constructions," in *Proc. IEEE INFOCOM*, 1999, pp. 708–716.

[3] C. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," *IEEE/ACM Trans. Networking*, vol. 8, no. 1, pp. 16–30, Feb. 2000.

[4] D. Balenson, D. McGrew, and A. Sherman, "Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization,", Internet Draft Report.

[5] S. Banerjee and B. Bhattacharjee, "Scalable secure group communication over IP multicast," *IEEE J. Select. Areas Commun.—Special Issue on Network Support for Group Communication*, vol. 20, no. 8, pp. 1511–1527, Oct. 2002.

[6] W. Trappe, J. Song, R. Poovendran, and K. J. R. Liu, "Key distribution for secure multimedia multicasts via data embedding," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 2001.

[7] R. Canetti, T. Malkin, and K. Nissim, "Efficient communication-storage tradeoffs for multicast encryption," *Eurocrypt*, pp. 456–470, 1999.

[8] G. Caronni, M. Waldvogel, D. Sun, and B. Plattner, "Efficient security for large and dynamic multicast groups," in *7th Workshop on Enabling Technologies (WET ICE '98)*, 1998.

[9] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[10] I. Ingemarsson, D. Tang, and C. Wong, "A conference key distribution system," *IEEE Trans. Inform. Theory*, vol. 28, no. 5, pp. 714–720, Sep. 1982.

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

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

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

[14] G. Ateniese, M. Steiner, and G. Tsudik, "New multiparty authentication services and key agreement protocols," *IEEE J. Select. Areas Commun.*, vol. 18, no. 4, pp. 628–639, Apr. 2000.

[15] V. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology: Crypto '85*, 1986, pp. 417–426.

[16] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proc. 7th ACM Conf. Computer Communication Security*, 2000, pp. 235–244.

[17] L. R. Dondeti, S. Mukherjee, and A. Samal, "DISEC: a distributed framework for scalable secure many-to-many communication," in *Proc. 5th IEEE Symp. Computers and Communications*, 2000, pp. 693–698.

[18] W. Trappe, Y. Wang, and K. J. R. Liu, "Group key agreement using divide-and-conquer strategies," presented at the The John's Hopkins University Conf. Information Sciences and Systems, Mar. 2001.

[19] ——, "Establishment of conference keys in heterogenous networks," in *Proc. IEEE Int. Conf. Communications*, 2002, pp. 2201–2205.

[20] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[21] A. Perrig, R. Szewczyk, D. Tygar, V. Wen, and D. Culler, "SPINS: security protocols for sensor networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, 2002.

[22] T. Cover and J. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[23] D. Huffman, "A method for the construction of minimum redundancy codes," in *Proc. Inst. Radio Eng.*, vol. 40, 1952, pp. 1098–1101.

[24] A. Turping and A. Moffat, "Practical length-limited coding for large alphabets," *Computer J.*, vol. 38, pp. 339–347, 1995.

[25] L. Larmore and D. Hirschberg, "A fast algorithm for optimal length-limited Huffman codes," *J. ACM*, vol. 37, pp. 464–473, Jul. 1990.

[26] R. Milidiu and E. Laber, "The warm-up algorithm: a Lagrangian construction of length restricted Huffman codes," *SIAM J. Computation*, vol. 30, pp. 1405–1426, 2000.

[27] B. Fox, "Discrete optimization via marginal analysis," *Manage. Sci.*, vol. 13, pp. 210–216, 1966.

[28] L. A. Wolsey, *Integer Programming*. New York: Wiley, 1998.

[29] B. Sun, W. Trappe, Y. Sun, and K. J. R. Liu, "A time-efficient contributory key agreement scheme for secure group communication," in *Proc. IEEE Int. Conf. Communications*, vol. 2, 2002, pp. 1159–1163.

[30] A. Law and W. Kelton, *Simulation Modeling and Analysis*, 2nd ed. New York: McGraw-Hill, 1991.

[31] T. Nemetz, "On the word-length of Huffman codes," *Probl. Contr. Inform. Theory*, vol. 9, pp. 231–242, 1980.

[32] E. Gilbert, "Codes based on inaccurate source probabilities," *IEEE Trans. Inform. Theory*, vol. 17, no. 3, pp. 304–314, May 1971.

[33] F. Fabris, A. Sgarro, and R. Pauletti, "Tunstall adaptive coding and miscoding," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 2167–2180, Nov. 1996.

Wade Trappe (M'02) received the B.A. degree in mathematics from The University of Texas at Austin in 1994, and the Ph.D. degree in applied mathematics and scientific computing from the University of Maryland, College Park, in 2002.

He is currently an Assistant Professor at the Wireless Information Network Laboratory (WINLAB) and the Electrical and Computer Engineering Department at Rutgers University. He is a co-author of the textbook *Introduction to Cryptography with Coding Theory* (Prentice Hall, 2001). His research interests include wireless network security, wireless networking, and multimedia security.

While at the University of Maryland, Dr. Trappe received the George Harhalakis Outstanding Systems Engineering Graduate Student Award. He is a member of the IEEE Signal Processing and Communication societies.

Yuke Wang received the Ph.D. degree in computer science from the University of Saskatchewan, Canada, in 1996. Dr. Yuke Wang

He is currently an Associate Professor with the University of Texas at Dallas. His research interests include network security, QoS, ASIC design and embedded processors for applications in DSP and communication systems.

Dr. Wang has served as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, the *EURASIP Journal of Applied Signal Processing*, and the *Journal of Circuits, Systems, and Signal Processing*.

K. J. Ray Liu (F'03) received the B.S. degree from the National Taiwan University in 1983, and the Ph.D. degree from the University of California at Los Angeles in 1990, both in electrical engineering.

He is a Professor in the Electrical and Computer Engineering Department and Institute for Systems Research of the University of Maryland, College Park. His research contributions encompass broad aspects of wireless communications and networking, information security, multimedia communications and signal processing, signal processing algorithms and architectures, and bioinformatics, in which he has published over 300 refereed papers.

Dr. Liu is the recipient of numerous honors and awards including IEEE Signal Processing Society 2004 Distinguished Lecturer, the 1994 National Science Foundation Young Investigator Award, the IEEE Signal Processing Society's 1993 Senior Award (Best Paper Award), IEEE 50th Vehicular Technology Conference Best Paper Award, Amsterdam, 1999, and EURASIP 2004 Meritorious Service Award. He also received the George Corcoran Award in 1994 for outstanding contributions to electrical engineering education and the Outstanding Systems Engineering Faculty Award in 1996 in recognition of outstanding contributions in interdisciplinary research, both from the University of Maryland. He is the Editor-in-Chief of *IEEE Signal Processing Magazine* and was the founding Editor-in-Chief of *EURASIP Journal on Applied Signal Processing*. He is on the Board of Governors and has served as Chairman of the Multimedia Signal Processing Technical Committee of the IEEE Signal Processing Society.