To appear in 23^{rd} *National Information Systems Security Conference*

An Efficient Secure Authenticated Group Key Exchange Algorithm for Large and Dynamic Groups*

Jim Alves-Foss
Center for Secure and Dependable Software
University of Idaho

Abstract

We present a new secure authenticated group key exchange algorithm for large groups. The protocol scales efficiently and performs well for dynamic group operations such as join, leave, merge and sub-grouping. The algorithm converts any underlying two-party key exchange algorithm into an efficient group key exchange algorithm.

1 Introduction

Diffie and Hellman (DH) introduced the concept of two-party key exchange in 1976 [7]. Since that time there have been several attempts to extend their efficient algorithm to multi-party situations, specifically [5, 8, 14, 16]. The importance of such algorithms can not be overstated. The ability to dynamically and publicly establish a session key for secure communication between a group of participants is a foundation of many secure group applications, such as conference calls, distributed computation, and distributed databases.

There are two classes of solutions that have been proposed for group key establishment in the literature. The first class is based on key distribution schemes where a single participant generates keying material and securely transmits it to all parties. If designed or implemented correctly this type of algorithm can be very efficient [6, 11, 17, 18], especially if it is developed hierarchically. However, this approach requires a trusted group controller to create and distribute the keys. All participants must assume that the generated keying matieral is valid and secure. Such approaches are not the focus of this paper and will not be discussed further.

The second class of solutions is based on contributory key exchange algorithms where all participants P_i generate a private value k_i and a corresponding public value PK_i. Through an exchange of public values and additional key exchange operations (KEOs) they reach agreement on a final shared key. These approaches maintain the character of DH key exchange in that all parties are involved in the generation of the new shared key.

In this paper, we present an efficient authenticated multi-party contributory key exchange algorithm that operates in the spirit of the DH algorithm and is based on work in [1, 4]. Our algorithm

*This work was supported in part by a grant from DOE.

does not require the use of a specific underlying two-party key exchange algorithm, but rather can be used with any underlying key exchange algorithm that uses public communication to establish the shared key. Such algorithms include DH-style two-party key exchange based on discrete logarithms [7], Galois field (2^k) [13], or Elliptic-Curves [12]. For groups of size n, our algorithm is efficient in that the number of sequential KEOs scales with $\log_2 n$ (unlike other algorithms that scale with n or even n^2.) The major benefits of contributory key exchange algorithms, such as the one presented here, include:

- all participants can independently calculate the same final key K, using public information PK_j (for $j \neq i$) and their own private key k_i.
- all participants can validate that their contribution k_i is included in the generation of K (thus guaranteeing that the freshness of K is at least that of k_i). For one of the algorithms presented in this paper (Alg1-DH), tests have shown that changing a single bit of one of the participant's initial private key results in a change of half of the bits in the final key, the same result we expect for randomly chosen initial private keys.
- the security of each k_i and K is maintained.

The remainder of this paper is organized as follows. We begin with an overview of two-party key exchange algorithms in Section 2. We then describe our general group key exchange algorithm in Section 3 and prove the security of the generated key from the DH variant of our protocol in Section 4. We then discuss the use of this algorithm in a dynamic group environment in Section 5. Next, in Section 6 we discuss the enhancement of the security of the algorithm, specifically in the context of authentication. We conclude the paper with a discussion of the protocol in the context of related work.

2 Two-Party Key Exchange Algorithms

Two-party public key exchange algorithms involve the exchange of public keys and the use of the partner's public key and a participant's private key to generate a new shared key. This section briefly describes two such algorithms. Note that these are not *authenticated* algorithms, in that although a shared secret is generated, the identity of the participants is not verified. A discussion of authenticated versions of this protocol is included in Section 6.

2.1 Diffie-Hellman Key Exchange

The DH algorithm [7] proceeds as depicted in Figure 1. The two participants agree on a large prime q and a generator $\alpha < q$. Each participant, $i \in \{0, 1\}$ then chooses a random private key k_i and generates a corresponding public key $PK_i = \alpha^{k_i} \texttt{mod} q$. The two participants then exchange public values and raise them to the power of their private key (for example, participant 1 generates $K = (PK_0)^{k_1} \texttt{mod} q$.) Both participants therefore generate the same shared key $K = \alpha^{k_0 k_1} \texttt{mod} q$. This key can then be used by each participant to independently generate the same session key, $k_s = f(K)$ for some predetermined secure function f. The Diffie-Hellman problem (DHP) states that it is computationally infeasible to determine K given (PK_0, PK_1, α, q). Although not proven, it is widely believed that DHP is true.

jointly select α and q
initialize k_i, a privately generated key
calculate $PK(i) = \alpha^{k_i} \texttt{mod} q$
transmit $PK(i)$
receive $Y = PK_{i \oplus 1}$ from partner
calculate $K = PK_{i \oplus 1}^{k_i} \texttt{mod} q = \alpha^{k_1 k_2} \texttt{mod} q$

Figure 1: Behavior of Participant i in the Diffie-Hellman Two-Party Key Exchange.

2.2 Elliptic-Curve Key Exchange

jointly select prime p, curve $E_p(a, b)$ and generator $G = (x_1, y_1) \in E_p(a, b)$.
initialize k_i, a privately generated key
calculate $PK(i) = k_i G$; a point in $E_p(a, b)$
transmit $PK(i)$
receive $Y = PK_{i \oplus 1}$ from partner
calculate $K = k_i P_{i \oplus 1}$

Figure 2: Behavior of Participant i in the Elliptic Curve Two-Party Key Exchange.

Following the style of DH key exchange, one can create a two-party key exchange algorithm using elliptic curves as depicted in Figure 2, a complete discussion can be found in [12]. The two participants agree on a large prime p, a curve $E_p(a, b)$ and a generator $G = (x_1, y_1) \in E_p(a, b)$ such that the smallest value of n for which $nG = O$ is a very large prime. Each participant then chooses a random private key k_i and generates a corresponding public key $PK_i = k_i G$. The two participants then exchange public values and multiply them with their private key (for example, participant 1 generates $K = k_1 P_0$.) Both participants therefore generate the same key $K = k_0 k_1 G$, which is actually a pair of numbers. At this point, each participant can independently calculate the same session key $k_s = f(K)$ for some predetermined f. The literature on elliptic curves suggests that it is infeasible to determine k given (G and kG) [12].

3 Efficient Authenticated Contributory Group Key Exchange

In this paper we present a new authenticated group key exchange protocol based on the security of two-party key exchange algorithms such as those discussed in the previous section. Our solution in this section is similar to that presented in [4, 1]; we enhance it with authentication in Section 6. Treating a group as an abstract entity we expand the traditional two-party key exchange algorithms into the group setting. With this new point of view we are able to build a new group, add members and merge groups through an iterative process using two-party key exchange as a primitive operation.

In the simplest case, assume that all group members are leaves of a complete binary tree, and each member begins the algorithm occupying its own group. Each group then executes an authenticated two-party key exchange with its sibling group, forming a new composite group defined

$G(i,r)$	group id for participant i in round r,	$=$	$\lfloor \frac{i}{2^{r-1}} \rfloor$
$C(g,r)$	id of the controller of group g in round r,	$=$	$g * 2^{r-1}$
$P(g,r)$	controller id for group $g's$ partner in round r,	$=$	$\begin{cases} C(g-1,r) & g \text{ is odd} \\ C(g+1,r) & g \text{ is even} \end{cases}$
$SK(i)$	current secret key for i. $SK(i)$ is initialized with some secret key for i.		
$PK(i)$	current public key for participant i.		
$GEN(k)$	generates a public value from a secret key, k.		
$TP(k,y)$	two-party key exchange, calculates a new secret key from a privately held key k and a public value y.		

Figure 3: Functions for Algorithm 1.

```
initialize SK(i) = k_i, a privately generated key
for  r = 1...log_2 n
     let g = G(i,r)
     calculate PK(i) = GEN(SK(i))
     if (i == C(g,r)) then
       transmit PK(i)
     endif
     receive Y = PK(P(g,r)) from partner
     calculate SK(i) = TP(SK(i),Y)
```

Figure 4: Behavior of participant i in Algorithm 1.

by the parent node. All group members update to the new shared group key. The process is repeated until there is only one group at the root of the tree. A more detailed description of this algorithm is given below. A mathematical analysis of this type of algorithm can be found in [4], with a proof of the efficiency of the approach.

3.1 Algorithm 1

Our basic group key exchange algorithm is used to create a new group from a collection of subgroups. In the basic description here, it is assumed that all group members know the identities of all other group members, and that there is a total ordering on the member identifiers (see Section 6 for a description of the algorithm using other topologies). We utilize a binary workload distribution to reduce the sequential execution time of the algorithm. The similarity of our approach to that of [4] suggests the security of the algorithm will hold up under scrutiny. In addition, a proof of the security of the DH variant of our algorithm is given in Section 4.

Assume that there are $n = 2^m$ group members, a two-party key exchange algorithm, and the functions defined in Figure 3. During the course of the algorithm, each participant P_i executes m rounds as depicted in Figure 4.

Initially, each participant selects a private key and is assigned to its own group. They then execute the rounds of Figure 4. The behavior of a round consists of two subgroups merging to form a new composite group using an underlying two-party key exchange algorithm. Each participant starts the round with a secret key, shared with all other subgroup members. All participants calculate a public key from this shared secret key (as they would in the two-party case); and the group controller broadcasts it to all members of the partner subgroup[1]. Each group member, upon receiving the partner's public key, calculates a new shared secret key for the new composite group (as it would in the two-party case). A new controller is selected for the composite group. The algorithm repeats until there is only one group.

Algorithm 1 executes in $\texttt{log}_2 n$ rounds, with each group member calculating at most a new group key and a new public key in each round. Therefore the sequential run time of this algorithm scales with the logarithm of the group size, unlike other published algorithms (see Section 8) which scale linearly (or worse) with group size. In addition, as we show in Section 4, the resulting key is secure.

3.2 Algorithm 1 Using Diffie-Hellman Key Exchange

As an example, this section defines Alg1-DH, a variant of Algorithm 1 which uses the DH algorithm as its underlying two-party key exchange algorithm. Alg1-DH defines $GEN(k) = \alpha^k$ and $TP(k, y) = y^k$, where all operations are performed mod$q$. The remaining functions remain as defined in Figure 3. The algorithm proceeds as follows:

1. Each participant, P_i, generates a DH key pair consisting of a private key k_i and a corresponding public key, $PK_i = \alpha^{k_i}$, where α and q are publicly agreed upon. Each P_i is then placed in its own group G_i and is considered controller of that group. The group G_i has private key k_i and public key PK_i.

2. For rounds $1 \ldots \texttt{log}_2 n$, we merge pairs of groups G_i and G_{i+1} (where $0 < i < n$ and i is odd), into a new group. This involves the use of the DH algorithm to calculate a new DH key pair with the private key $k = \alpha^{k_{G_i} k_{G_{i+1}}}$ and public key $PK = \alpha^k$. Every member of the new group shares this key pair.

3. The resulting new group designates a specific member to be the group controller. The group controllers are responsible for exchanging key generation messages. Although not required by this algorithm, it is possible that some system architectures may require additional functionality of the controller such as authentication, message routing or other group management functions.

At the end of this algorithm, every member shares the same DH group key, since each group merge is effectively a standard DH key exchange. The proof of this point is straightforward and is based on the fact that all members of the same subgroup share the same DH key pair, which can now be used to generate session keys as define in Section 2. For example, an 8 member group would generate the final DH group key ($\texttt{mod}q$) as:

$$\alpha^{\left(\alpha^{\left(\alpha^{(k_1 k_2)}\alpha^{(k_3 k_4)}\right)}\alpha^{\left(\alpha^{(k_5 k_6)}\alpha^{(k_7 k_8)}\right)}\right)}$$

[1] In [4] the authors propose having each group member transmit a single message to an equivalent member in the partner group, eliminating the need for broadcasts.

Algorithm 1 works equally well with other two-party key exchange algorithms such as those based on Elliptic Curves [12], or hybrid combinations of exchange algorithms, where different subgroups internally use different key exchange algorithms. Section 5 outlines the proposed use of this new algorithm in various group management situations. However, we will first prove the security of Alg1-DH.

4 Security of Algorithm 1

Assuming the security of the DH algorithm, we show that Alg1-DH is secure. In this proof, all operations are assumed to be performed $\mathtt{mod} q$.

Assumption: Let k_1 and k_2 be random numbers, and $PK_i = \alpha^{k_i}$ be the corresponding public derived values. The shared value $k = \alpha^{PK_1 PK_2}$ derived using the two-party Diffie-Hellman key exchange algorithm is indistinguishable (in polynomial time) from a random number.

Theorem 1. *Let G be a group of size $n = 2^m$ and let $k_1 \dots k_n$ be random numbers. The shared value k derived in the application of Alg1-DH using initial values $k_1 \dots k_n$ is indistinguishable (in polynomial time) from a random number.*

Proof. The proof is based on induction over m.

Base Case: $m = 1$. This defaults to the two party Diffie-Hellman key exchange, which is assumed to be secure.

Induction Hypothesis: The shared value k derived in the application of Alg1-DH, for any group of size $2^m (1 \leq m)$ is indistinguishable (in polynomial time) from a random number.

Induction Step: Alg1-DH creates a group of size 2^{m+1} by merging two subgroups of size 2^m. Let k_1 and k_2 denote the shared secret values of these two subgroups. Alg1-DH uses these shared values to create two public values $PK_i = \alpha^{k_i}$, and then executes the two-party DH to generate the new shared key $k = \alpha^{k_1 k_2}$. From the induction hypothesis, k_1 and k_2 are indistinguishable from random values. From the DH assumption, the value k is therefore indistinguishable from a random number.

□

Theorem 1 immediately leads to the corollary that the key generated from the merger of two binary-sized groups (where group size is a power of 2) is secure.

Corollary 1. *Let G_1 and G_2 be two groups of size $n_1 = 2^{m_1}$ and $n_2 = 2^{m_2}$ respectively, with secure shared group keys. The shared value k derived from the merger of these groups using Alg1-DH is indistinguishable (in polynomial time) from a random number.*

Proof. Let k_1 and k_2 denote the shared secret values of these two subgroups. Alg1-DH uses these shared values to create two public values $PK_i = \alpha^{k_i}$, and then executes the two-party DH to generate the new shared key k. Given that k_1 and k_2 are secure (indistinguishable from random values) and the DH assumption, the value k is therefore indistinguishable from a random number.

□

In general, we do not have to limit ourselves to binary sized groups. We can break down any size group into a collection of binary sized groups and merge those. This is a straight-forward extension of algorithm 1 as discussed in Section 6.

Theorem 2. *Let G be a group of size n and let $k_1 \ldots k_n$ be random numbers. The shared value k derived in the application of Alg1-DH using initial values $k_1 \ldots k_n$ is indistinguishable (in polynomial time) from a random number.*

Proof. This proof is based on the binary decomposition of the size of the group.

Case 1: If $n = 2^m$ for some integer m, then we have the same situation as Theorem 1 and we are done.

Case 2: If $n \neq 2^m$ for any integer m, we can rewrite n as the sum of its binary decomposition $n = x_m 2^m + x_{m-1} 2^{m-1} + \ldots + x_1 2^1 + x_0 2^0$ where each $x_i \in 0, 1$. A generic Alg1-DH will create these groups first before merging them together. Each of these group's key is secure (as per Theorem 1). Merging pairs of these groups will result in the same two-party interactions we saw in Corollary 1. Therefore the merged groups have secure shared keys. We repeat this process until there is only one group with a secure key k.

□

5 Dynamic Group Operations

Several groupware products require dynamic group behavior. This includes the ability to add and delete group members, to combine groups, and to create subgroups. In this section we discuss the efficient use of Algorithm 1 to support these operations.

5.1 Group Genesis

Group genesis involves the initial creation of the group. As with other group management algorithms, it is required that at least one member (the group controller) have prior knowledge of the group members with some ordered labeling scheme. This information is then sent to all other group members (if this is not desired, the controller can either route necessary key exchange messages or pass on member identifiers on a need-to-know basis). Using this labeling scheme we logically arrange group members such that they can self select groups of size 2^i ($i = 0 \ldots \mathtt{log}_2 n$) for stage i of the algorithm. We then proceed with algorithm 1.

5.2 Group Fusion

Group fusion is a basic operation in algorithm 1. As such, any subsequent group fusion can be implemented as if it was the final stage of algorithm 1 and therefore requires only one KEO for every member in the new composite group (these can be executed in parallel). The group controllers may store the previous keys to support security policies that permit re-keying when some group members are off-line, or permit reuse of old keys for re-join.

5.3 Single Member Join

A single member join is a special case of group fusion (with a group sizes of n and 1). The algorithm executes as specified above.

5.4 k-Member Join

If the group controller receives a join request from k independent entities, it can treat them as k separate single member joins, or it can off-load the workload by requesting the k entities to form their own independent group. Subsequent to that operation (which takes $\log_2 k$ rounds) the new group controller will initiate group fusion with the original group.

5.5 Sub-grouping, Single-Member Leaves and k-Member Leaves

At this point we have not determined any more efficient algorithm to subgrouping or leave operations other than a complete re-execution of algorithm 1. However, in the case of a single-member leave, only those groups that have been reformed (due to the loss of the member) need to recalculate their group key pairs. This means that in each iteration of the algorithm we double the number of members involved in the calculation, reducing the total computation and communication cost but not the number of rounds. Even without more efficient algorithms, a complete re-run of algorithm 1 is sequentially more efficient than published subgrouping algorithms [15].

6 Authenticated Group Key Exchange

Authenticated key exchange involves authenticating the identity of group members. Based on group security policy this may involve anything from authentication of the new member to a single exiting group member to authentication to every exiting group member. It is assumed that it is acceptable for this authentication to utilize public-key authentication techniques. Some systems may require authentication only to a central controller, others may permit implicit authentication during group merge, and others may require complete pair-wise authentication as discusses below. Several approaches to authenticated group key exchange have been discussed in the literature [3, 6, 9, 10]. A full analysis of the methods presented here is left to subsequent publications.

6.1 Centralized Authentication – Algortihm 2

A group may designate a single trusted controller to perform authentication for the whole group. In this situation, authentication could occur at the start of the key exchange algorithm. Each participant could transmit its initial public key PK_i using an authenticated message. The controller would then validate the authenticity of the senders and then send a message to all group members specifying the identities of the authenticated users. The algorithm would then proceed as normal, using the authenticated public keys. To enhance security, the authentication of users could be repeated at every round, authenticating all of the key exchange messages. Or, a third option is to have each participant create a new authenticated message that contains a value encrypted with the new shared key. An outline of this algorithm is demonstrated in Figure 5.

```
initialize SK(i) = k_i, a privately generated key
calculate PK(i) = GEN(SK(i))
transmit signed = PK(i) to central authority
receive list of authenticated participants from central authority
for  r = 1...log_2 n
     let g = G(i, r)
     calculate PK(i) = GEN(SK(i))
     if (i == C(g, r)) then
       transmit PK(i)
     endif
     receive Y = PK(P(g, r)) from partner and checks identity
     calculate SK(i) = TP(SK(i), Y)
```

Figure 5: Behavior of participant i in Algorithm 2.

```
initialize SK(i) = k_i, a privately generated key
for  r = 1...log_2 n
     let g = G(i, r)
     calculate PK(i) = GEN(SK(i))
     if (i == C(g, r)) then
       transmit authenticated/signed PK(i)
     endif
     receive Y = PK(P(g, r)) and checks authentication
     calculate SK(i) = TP(SK(i), Y)
```

Figure 6: Behavior of participant i in Algorithm 3.

6.2 Implicit Authentication – Algorithm 3

We define *implicit authentication* as the process of authenticating the group controller and then trusting the controller to have authenticated group members. This approach directly maps to authenticated two-party key exchange algorithms where the participants authenticate the public values distributed by their partners. In the group case, all members of a group would authenticate the message from the controller (or their partner in the BW case [4]) of the other group. The key exchange messages could include the identity of other group member. In either case the users will implicitly assume the authentication of the other group's members. An outline of this algorithm is demonstrated in Figure 6.

6.3 Pairwise Authentication – Algorithm 4

At the end of Algorithm 1, we have a group of users with a shared key. We can have each member encrypt a message with the shared key and then broadcast (multicast) it in an authenticated message to all group members. The other members would then authenticate each group member's identity. If any of the authentications fail, the group key is discarded and an new key is generated

```
Perform Algorithm 1 resulting in new group key K
Generate new random value, v.
Generate c_i = v_K
Broadcast authenticated/signed c_i to all group members
Receive and verify all c_j where j ≠ i
```

Figure 7: Behavior of participant i in Algorithm 4.

without the unauthenticated member. An outline of this algorithm is demonstrated in Figure 7.

7 Variants

In this section we discuss some possible variations and uses of Algorithm 1 on different network topologies. This section is only meant to provide a brief overview of these concepts, specific details are left to a future paper.

Algorithm 1 was defined in terms of a perfectly balanced binary tree, with each participant knowing the identities of all other participants. There are variations to this configuration which need to be addressed. Becker and Willie provide a theoretical foundation for the minimal number of messages exchanged when the number of participants $\neq 2^n$ in a fully connected environment, but do not address other topological limitations [4].

Partial Binary Trees In the description of Algorithm 1, we assumed that we had $n = 2^m$ participants, such that we could map them to a complete binary tree. However, the algorithm can be easily tailored for any number of members n, where $2^m < n < 2^{m+1}$.

In one approach, we have participant P_n represent pseudo participants $n+1\ldots, 2^{m+1}$. Each of the pseudo participants has private key 1, and public key α (the same is true of any group consisting of only pseudo participants). P_n will broadcast these values when necessary.

In another approach, we divide the group into several subgroups, each with a group size a power of 2. This can be accomplished by the binary decomposition $n = x_m 2^m + x_{m-1}2^{m-1} + \ldots + x_1 2^1 + x_0 2^0$ where each $x_m \in \{0, 1\}$. Each of the groups is formed, and then pairs of the groups are merged until one group remains.

A Linked List Let us assume that participants information is originally organized in a bi-directional linked list, with adjacent group controllers knowledgeable about their immediate neighbors (recall that groups are initialized with one participant/controller per group). In the first round, pairs of neighbors exchange values and one is chosen as a group controller for the merged group. In addition, all participants pass on neighbor information. In subsequent rounds, we use the passed on neighbor information to determine the identity of the new controller for the neighboring group. In the round, keys and group neighbor information is exchanged, in addition the controller must propagate key information down through its group (this can be accomplished along the original linked list). In each round our neighbor information doubles in distance, allowing a logarithmic execution of the algorithm.

8 Comparison

There are two measures of cost in key distribution. The first, based on message transmissions, has been examined in [4]. The second is based on the cost of computations. A single two-party key exchange operation (KEO) such as exponentiation is used as the basis of the computation cost of group key exchange algorithms. Of the published algorithms analyzed in [16] the number of sequential KEOs performed scale at least linearly with group size. Hence a doubling of the group size doubles the time it takes to complete the algorithm. Our algorithm scales logarithmically with group size, so a doubling of the group results in only 1 additional sequential KEOs. Since the KEOs are computationally expensive, our algorithm provides a tremendous cost savings over other proposed algorithms by greatly reducing this computational workload by distributing the work and the key generation. Specifically we require on the order of $\log_2 n$ sequential exponentiations with a total of $n\log_2 n$ total exponentiations. This is a substantial savings in time and total computational workload over previous work cited above. For comparison purposes the performance of algorithm 1 for group genesis is summarized below, a detailed discussion of the performance of other algorithms can be found in [4, 16].

rounds	$\log_2 n$
messages	$2n$
combined message size	$2n$
exponentiations per P_i	$\leq 2 + 2\log_2 n$
sequential exponentiations	$2 + 2\log_2 n$
total exponentiations	$\leq n(2 + 2\log_2 n)$

In addition to the savings in group genesis, member addition in algorithm 1 requires only one KEO, as efficient as any of the other algorithms. Addition of k members operates as a separate group genesis for k members and then a single KEO for the final merge, this is more efficient than the other algorithms. The only place where algorithm 1 may fall back is in removal of a single member or small number of members. Our algorithm requires a complete rerun of the genesis algorithm, where the IKA.1 and IKA.2 algorithms can perform this operation rather efficiently [15] (although they still require n sequential exponentiations).

9 Conclusions

In this paper we presented a secure authenticated group key exchange algorithm that operates efficiently for large groups. Unlike other algorithms presented in the literature, our algorithm does not scale linearly with group size, but logarithmically; resulting in tremendous performance improvements for large groups. The only other contributory algorithm that performs this well is the Becker-Willie algorithm in [4]. We have extended this work to include authentication. The major benefits of this algorithm can be summarized as follows:

- The algorithm run time is logarithmic in the size of the group.
- It is a true contributory key exchange algorithm.
- At each stage of the algorithm, the group merge can utilize any underlying two-party key exchange algorithm, and can securely redefine the group shared key.

- It is a provable secure algorithm.
- We have demonstrated authentication in this algorithm.

The algorithm, as presented, does not provide for a variation in network topologies. Although this issue was discussed, it was felt that a detailed discussion be left for a subsequent publication. In addition, we do not address issues related to the correct selection of parameters for two-party key exchange, but assume that these are addressed according to good practices [2, 10].

References

[1] J. Alves-Foss. An efficient secure group key exchange algorithm for large and dynamic groups. Technical Report CSDS-99-08, Center for Secure and Dependable Software, Univ. of Idaho, 1999.

[2] R. Anderson and S. Vaudenay. Minding your p's and q's. In *Advances in Cryptology – ASIACRYPT '96*, pages 26–35, 1996.

[3] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. Authenticated group key agreement and friends. In *5th ACM Conference on Computer and Communications Security*, pages 17–26, San Francisco, California, November 1998. ACM Press.

[4] K. Becker and U. Willie. Comunication complexity of group key distribution. In *5th Conference on Computers & Communication Security*, pages 1–6, 1998.

[5] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology - EUROCRYPT'94*, pages 275–286, May 1994.

[6] M. Burmester and Y. Desmedt. Efficient and secure conference key distribution. In *Security Protocols: International Workshop*, pages 119–129, April 1996.

[7] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–652, Nov. 1976.

[8] I. Ingemarsson, D. Tang, and C. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, Sep. 1982.

[9] M. Just and S. Vaudenay. Authenticated multi-party key agreement. Technical Report SCS-TR-96-04, Carleton University, Computer Science Department, Ottowa CA, 1996.

[10] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Advances in Cryptology – ASIACRYPT '96*, pages 36–49, 1996.

[11] D. McGrew and A. Sherman. Key establishment in large dynamic groupd using one-way function trees. Submitted to IEEE Transactions on Software Engineering, May 1998.

[12] A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[13] S. Pohlig and M. Hellman. An imporved algortihm for computing logarithms in $gf(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–111, January 1978.

[14] D. Steer, L. Sawczynski, W. Diffie, and M. Weiner. A secure audio teleconference system. In *Advances in Cryptology - CRYPTO'88*, Aug. 1990.

[15] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. Technical report, Information Sciences Institute, Jan. 1999.

[16] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-hellman key distribution extended to groups. In *Third ACM Conference on Computer and Communications Security*, pages 31–37. ACM Press, March 1996.

[17] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. Internet Draft (Work in progress), July 1998.

[18] C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. In *Proc. of ACM SIGCOMM '98 Conference on Applications, technologies, architectures and protocols for comptuer communications*, pages 68–99, 1998.