An Efficient Tree-based Group Key Agreement using Bilinear Map

Sangwon Lee[1], Yongdae Kim[2], Kwangjo Kim[1], and Dae-Hyun Ryu[3]

[1] Information and Communications University (ICU),
58-4, Hwaam-Dong, Yuseong-gu, Daejeon, 305-732, Korea,
{swlee,kkj}@icu.ac.kr
[2] University of Minnesota - Twin cities
4-192 EE/CSci Building, 200 Union Street S.E., Minneapolis, MN 55455
kyd@cs.umn.edu
[3] Hansei University,
604-5, Dangjung-dong, Kunpo-si, Kyunggi-do, Seoul, 435-742, Korea,
dhryu@hansei.ac.kr

Abstract. Secure and reliable group communication is an increasingly active research area by growing popularity in group-oriented and collaborative application. One of the important challenges is to design secure and efficient group key management. While centralized management is often appropriate for key distribution in large multicast-style groups, many collaborative group settings require distributed key agreement. The communication and computation cost is one of important factors in the group key management for Dynamic Peer Group. In this paper, we extend TGDH (Tree-based Group Diffie-Hellman) protocol to improve the computational efficiency by utilizing pairing-based cryptography. The resulting protocol reduces computational cost of TGDH protocol without degrading the communication complexity.

Keywords: Group key agreement, TGDH, Bilinear Diffie-Hellman, Bilinear map, Pairings

1 Introduction

Secure and reliable communications have become critical in modern computing. The centralized services like e-mail and file sharing can be changed into distributed or collaborated system through multiple systems and networks. Basic cryptographic requirements such as data confidentiality, data integrity, authentication and access control are required to build secure collaborative system in the broadcast channel. When all group members have the shared secret key, these security services can be easily implemented.

Dynamic Peer Group (DPG) belongs to a kind of *ad hoc* group which its membership can be frequently changed and the communicating party in a group can be dynamically configured.

Recently, Joux[5] presented a three-party key agreement protocol which requires each entity to make on a single round using pairings on algebraic curves.

This should be contrasted with the obvious extension of the conventional Diffie-Hellman key distribution protocol to three parties requiring two interactions per peer entity. We extend this three-party key agreement protocol to group key agreement protocol using ternary tree and also use two-party key agreement protocol for some subtree node.

Y. Kim *et al.*[9] proposed a secure, simple and efficient key management method, called TGDH(Tree-based Group Diffie-Hellman) protocol, which uses key tree with Diffie-Hellman key exchange to efficiently compute and update group keys. Since the computation cost of tree-based key management is proportional to the height of configured key tree. Using ternary key tree, we can reduce the computation cost $O(log_2 n)$ of TGDH to $O(log_3 n)$.

This paper is organized as follows: Section 2 briefly introduces previous work in group key management, group membership events and bilinear map. Section 3 explains the protocol. Performance analysis is described in Section 4. We suggest concluding remarks in Section 5 following with the security analysis of our protocol in Appendix.

2 Previous Work

2.1 Group Membership Operations

A comprehensive group key agreement must handle adjustments to group secrets subsequent to all membership operations in the underlying group communication system.

We distinguish among single and multiple member operations. Single member changes include member addition or deletion. This occurs when a member wants to join(or leave) a group. Multiple member changes also include addition and deletion: ***Member Join*** and ***Leave***. We refer to the multiple addition operation as ***Group Merge***, in which case two or more groups merge to form a single group. We refer to the multiple leave operation as ***Group Partition***, whereby a group is split into smaller groups. ***Group Merge*** and ***Partition*** event are common owing to network misconfiguration and router failures. Hence, dealing with ***Group Partition*** and ***Merge*** is a crucial component of group key agreement.

In addition to the single and multiple membership operations, periodic refreshes of group secrets are advisable so as to limit the amount of ciphertext generated with the same key and to recover from potential compromise of member's contribution or prior session keys. ***Key Refresh*** is one of the most important security requirements of a group key agreement.

The special member, referred to as *sponsor*, is responsible for broadcasting all link values of the current tree to the members. Note that the *sponsor* is not a privileged member. His task is only to broadcast the current tree information to the group members. Any current member could perform this task. We assume that every member can unambiguously determine both the *sponsors* and the insertion location in the key tree. ***Key Refresh*** operation can be considered to be a special case of ***Member Leave*** without any members actually leaving the group.

Group key agreement of dynamic group must provide four security properties: Group key secrecy is basically supported property in group communication. Forward secrecy means that any leaving member from a group can not generate new group key. Backward secrecy means that any joining member into a group can not discover previously-used group key. The combination of backward secrecy and forward secrecy forms key independence.

2.2 Bilinear Pairings and BDH Assumption

Let G_1 be an additive group generated by P, whose order is a prime q, and G_2 be a multiplicative group of the same order q. We assume that the discrete logarithm problem(DLP) in both G_1 and G_2 is hard. Let $e : G_1 \times G_1 \rightarrow G_2$ be a paring which satisfies the following conditions:

1. Bilinear: $e(P_1+P_2, Q) = e(P_1, Q)e(P_2, Q)$ and $e(P, Q_1+Q_2) = e(P, Q_1)e(P, Q_2)$
2. Non-degenerate : The map does not send all pairs in $G_1 \times G_1$ to the identity in G_2. Observe that since G_1, G_2 are groups of prime order this implies that if P is a generator of G_1 then $\widehat{e}(P, P)$ is a generator of G_2.
3. Computability : There is an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in G_1$

The Weil or Tate pairings associated with supersingular elliptic curves or Abelian varieties can be modified to create such bilinear maps.

BDH Problem : The Bilinear Diffie-Hellman(BDH) Problem for a bilinear map $e : G_1 \times G_1 \rightarrow G_2$ is defined as follows: given $P, aP, bP, cP \in G_1$, compute $e(P, P)^{abc}$, where a, b, c are randomly chosen from Z_q^*. An algorithm $\mathcal{A}$ is said to solve the BDH problem with an advantage of ϵ if

$$Pr[\mathcal{A}(P, aP, bP, cP) = e(P, P)^{abc}] \geq \epsilon$$

BDH Assumption : We assume that the BDH problem is hard, which means there is no polynomial algorithm to solve BDH problem with non-negligible probability.

3 Our Protocol

Table 1 shows the notations used in this paper. We can classify three nodes of a key tree as follows:

- Member node : represent each group member as leaf node.
- Key node : correspond with one key. This key is shared by all members of the subtree rooted at this key node.
- Root node : represent the shared group key.

Table 1. Notations

N	Member of protocol parties(group members)
C	Set of current group members
L	Set of leaving members
M_i	i-th group member; $i \in$ {1,2, ..., N}
h	The height of the key tree
$< l, v >$	v-th node at the l-th level in a tree
T_i	M_i's view of the key tree
$\widehat{T}_i$	M_i's modified tree after membership operation
$T_{<i,j>}$	A subtree rooted at node $< i, j >$
BK_i^*	set of M_i's blinded keys
P	Public information, a point on an elliptic curve
H_1	Hash function, $H_1 : G_2 \rightarrow Z_q^*$
H_2	Hash function, $H_2 : G_1 \rightarrow Z_q^*$

Fig. 1 shows an example of a key tree. The root is located at the 0-th level and the lowest leaves are at the h-th level. Since we use ternary tree, every node can be a leaf or a parent of two nodes or a parent of three nodes. The node are denoted $< l, v >$, where $0 \leq v \leq 3^l - 1$ since each level l hosts at most 3^l nodes. Each node $< l, v >$ is associated with the *key* $K_{<l,v>}$ and the blinded key (*bkey*) $BK_{<l,v>} = K_{<l,v>}P$. The multiplication kP is obtained by repeating k times addition over an elliptic curve. We assume that a leaf node $< l, v >$ is associated with M_i, then the node $< l, v >$ has M_i's session random key $K_{<l,v>}$. We further assume that the member M_i at node $< l, v >$ knows every key along the path from $< l, v >$ to $< 0, 0 >$, referred to as the *key-path*. In Fig. 1, if a member M_3 owns the tree T_3, then M_3 knows every *key* $\{K_{<2,2>}, K_{<1,0>}, K_{<0,0>}\}$ and every *bkey* $BK_3^* = \{BK_{<2,2>}, BK_{<1,0>}, BK_{<0,0>}\}$ on T_3.

The case of subtree having three child node at $< l, v >$, computing a key requires the knowledge of the *key* in one of the three child node and the *bkey* of the other child node. We can get a *key* $K_{<l,v>}$ by computing pairings. In another case, we need to know the *key* of one of the two child node and the *bkey* of the other child node. We can get a *key* $K_{<l,v>}$ by computing a point multiplication on elliptic curve. $K_{<0,0>}$ at the root node is the group secret shared by all members.

For example, in Fig. 1, M_3 can compute $K_{<1,0>}$, $K_{<0,0>}$ using $BK_{<2,0>}$, $BK_{<2,1>}$, $BK_{<1,1>}$ and $K_{<2,2>}$. The final group key $K_{<0,0>}$ is :

$$K_{<0,0>} = H_1(\widehat{e}(H_1(\widehat{e}(P,P)^{r_4 r_5 r_6})P, r_7 P)^{H_1(\widehat{e}(r_1 P, r_2 P)^{r_3})})$$

If there are 8 members in group, then the final group key $K_{<0,0>}$ is :

$$K_{<0,0>} = H_1(\widehat{e}(H_1(\widehat{e}(P,P)^{r_4 r_5 r_6})P, H_2(r_7 r_8 P)P)^{H_1(\widehat{e}(r_1 P, r_2 P)^{r_3})})$$

where $r_7 r_8 P$ is the shared key between M_7 and M_8 using ECDH (Elliptic Curve Diffie-Hellman)problem.

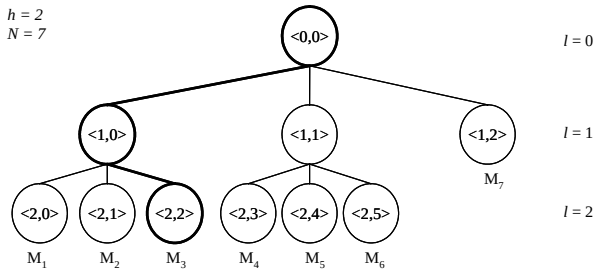

Fig. 1. An example of a key tree

Now we describe the group operation: ***Join***, ***Leave***, ***Partition*** and ***Merge***. We modify this operation in TGDH by utilizing the ternary tree and bilinear map.

3.1 Join Protocol

We assume the group has n members: $\{M_1, M_2, ..., M_n\}$. The new member M_{n+1} initiates the protocol by broadcasting a join request message that contains its own *bkey* $BK_{<0,0>}$ $(= r_{n+1}P)$.

Each current member receives this message and first determines the insertion point in the tree. The insertion point is the shallowest rightmost node, where the join does not increase the height of the key tree. Otherwise, if the key tree is fully balanced, the new member joins to the root node. The *sponsor* is the rightmost leaf in the subtree rooted at the insertion point. If the intermediate node in the rightmost has two member nodes, the *sponsor* inserts the new member node under this intermediate node. The tree becomes fully balanced. Otherwise, each member creates a new intermediate node and a new member node, and promotes the new intermediate node to be the parent of both the insertion node and the new member node. After updating the tree, all members, except the *sponsor*, are blocked. The *sponsor* proceeds to update his share and computes the new group key; the *sponsor* can do this operation since it knows all necessary *bkeys*. Next, the *sponsor* broadcasts the new tree which contains all *bkeys*. All other members update their trees accordingly and compute the new group key.

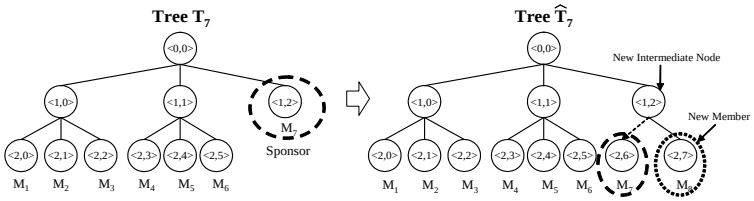

Fig. 2. Tree-updating in join operation

Table 2. Join Protocol

Step 1 : The new member broadcasts request for join

$$M_{n+1} \xrightarrow{BK_{<0,0>}=r_{n+1}P} C$$

Step 2 : Every member

– if key tree contains the subtree that has two child node, add the new member node for updating key tree. otherwise, add the new member node and new intermediate node,

– remove all *keys* and *bkeys* from the leaf node related to the *sponsor* to the root node.

The *sponsor* M_s additionally

– generates new share and computes all $[key, bkey]$ pairs on the *key-path*,

– broadcasts updated tree $\widehat{T}_s$ including only *bkeys*.

$$M_s \xrightarrow{\widehat{T}_s(BK_s^*)} C \cup \{M_{n+1}\}$$

Step 3 : Every member computes the group key using $\widehat{T}_s$.

It might appear wasteful to broadcast the entire tree to all members, since they already know most of the *bkeys*. However, since the *sponsor* needs to send a broadcast the entire tree to the group anyhow, it might as well include more information which is useful to the new member, thus saving one unicast message to the new member (which would have to contain the entire tree).

Fig. 2 illustrates an example of member M_8 joining a group where the *sponsor* (M_7) performs the following actions:

1. Rename node $< 1, 2 >$ to $< 2, 6 >$.
2. Generate a new intermediate node $< 1, 2 >$ and a new member node $< 2, 7 >$.
3. Update $< 1, 2 >$ as the parent node of $< 2, 6 >$ and $< 2, 7 >$.
4. Generate new share and compute all $[key, bkey]$ pairs.
5. Broadcast updated tree $\widehat{T}_7$.

Since all members know $BK_{<2,7>}$, $BK_{<1,0>}$ and $BK_{<1,1>}$, M_7 can compute the new group key $K_{<0,0>}$. Every other member also performs steps 1 and 2, but cannot compute the group key in the first round. Upon receiving the broadcasted *bkeys*, every member can compute the new group key.

If another member M_9 wants to join the group, the new *sponsor*(M_8) performs the following actions:

1. Generate a new member node $< 2, 8 >$ under the intermediate node $< 1, 2 >$.
2. Generate new share and compute all $[key, bkey]$ pairs.
3. Broadcast updated tree $\widehat{T}_8$.

Every member also performs step 1, and then can compute the new group key with the broadcasted messages.

3.2 Leave Protocol

Such as ***Join*** protocol, we start with n members and assume that member M_d leaves the group. The *sponsor* in this case is the rightmost leaf node of the subtree rooted at leaving member's sibling node. First, if the number of leaving member's sibling node is two, each member updates its key tree by deleting the leaf node corresponding to M_d. Then the former sibling of M_d is updated to replace M_d's parent node. Otherwise each member only deleting the leaf node corresponding to M_d. The *sponsor* generates a new key share, computes all $[key, bkey]$ pairs on the *key-path* up to the root, and broadcasts the new set of *bkey*. This allows all members to compute the new group key. In Fig. 3, if member M_7 leaves the group, every remaining member deletes $< 1, 2 >$ and $< 2, 6 >$. After updating the tree, the *sponsor* (M_{10}) picks a new share $K_{<2,8>}$, recomputes $K_{<1,2>}$, $K_{<0,0>}$, $BK_{<2,8>}$ and $BK_{<1,2>}$, and broadcasts the updated tree $\widehat{T}_{10}$ with BK^*_{10}. Upon receiving the broadcast message, all members compute the group key. Note that M_7 cannot compute the group key, though he knows all the *bkeys*, because his share is no longer a part of the group key.

Table 3. Leave Protocol

Step 1 : Every member
- update key tree by removing the leaving member node,
- remove relevant parent node, if this node have only one member node,
- remove all *keys* and *bkeys* from the leaf node related to the *sponsor* to the root node.

The *sponsor* M_s additionally
- generates new share and computes all $[key, bkey]$ pairs on the *key-path*,
- broadcasts updated tree $\widehat{T}_s$ including only *bkeys*.

$$M_s \xrightarrow{\widehat{T}_s(BK^*_s)} C - L$$

Step 2 : Every member computes the group key using $\widehat{T}_s$.

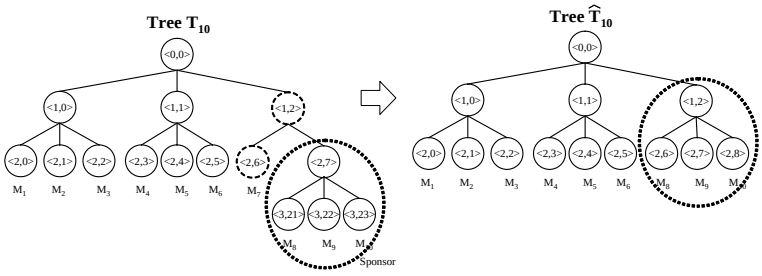

Fig. 3. Tree-updating in leave operation

In Fig. 3, if member M_{10} leaves the group, every remaining members delete only $< 3, 23 >$. After updating the tree, the *sponsor* (M_9) generates new share $K_{<3,22>}$, recomputes $K_{<2,7>}$, $K_{<1,2>}$, $K_{<0,0>}$, $BK_{<2,7>}$ and $BK_{<1,2>}$, and broadcasts the updated tree $\widehat{T}_9$ with BK_9^*. Upon receiving the broadcast message, all members can compute the group key.

3.3 Partition Protocol

We assume that a network failure causes a partition of the n-member group. From the viewpoint of each remaining member, this event appears as a simultaneous leaving of multiple members. The ***Partition*** protocol is involves multiple rounds; it runs until all members compute the new group key. In the first round, each remaining member updates its tree by deleting all partitioned members as well as their respective parent nodes and "compacting" the tree. The procedure is summarized in Table 4.

Table 4. Partition Protocol

Step 1 : Every member
- update key tree by removing all the leaving member node,
- remove their relevant parent node, if this node have only one member node,
- remove all *keys* and *bkeys* from the leaf node related to the *sponsor* to the root node.

Each *sponsor* M_{s_t}
- if M_{s_t} is the shallowest rightmost *sponsor*, generate new share,
- compute all $[key, bkey]$ pairs on the *key-path* until it can proceed,
- broadcast updated tree $\widehat{T}_{s_t}$ including only *bkeys*.

$$M_{s_t} \xrightarrow{\widehat{T}_{s_t}(BK^*_{s_t})} C - L$$

Step 2 to h (Until a *sponsor* M_{s_j} could compute the group key)
: For each *sponsor* M_{s_t}
- compute all $[key, bkey]$ pairs on the *key-path* until it can proceed,
- broadcast updated tree $\widehat{T}_{s_t}$ including only *bkeys*.

$$M_{s_t} \xrightarrow{\widehat{T}_{s_t}(BK^*_{s_t})} C - L$$

Step $h + 1$: Every member computes the group key using $\widehat{T}_s$.

Fig. 4 shows an example. In the first round, all remaining members delete all nodes of leaving members and compute *keys* and *bkeys*. Any member can not compute the group key since they lack the *bkey* information. However, M_5

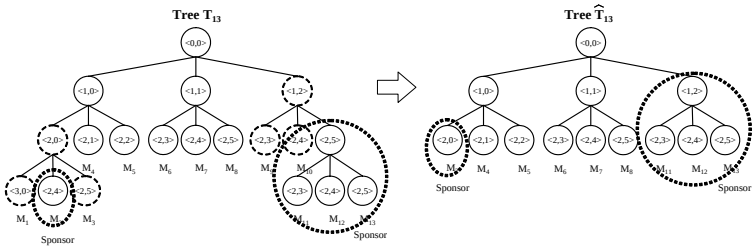

Fig. 4. Tree-updating in partition operation

generates new share and computes and broadcasts $BK_{<1,0>}$ in the first round, and M_{13} can thus compute the group key. After M_{13} generates new share and broadcasts $BK_{<1,2>}$, M_5 can compute the group key. Finally every member knows all *bkeys* and can compute the group key.

Note that if some member M_i can compute the new group key in round h', then all other member can compute the group key, in round $h' + 1$, since M_i's broadcast message contains every *bkey* in the key tree, each member can detect the completion of the partition protocol independently.

3.4 Merge Protocol

After the network failure recovers, subgroup may need to be merged back into a single group. We now describe the merge protocol for k merging groups.

In the first round of the merge protocol, each *sponsor*(the rightmost member of each group) broadcasts its tree with all *bkeys* to all other groups after updating the secret share of the *sponsor* and relevant $[key, bkey]$ pairs up to the root node. Upon receiving these message, all members can uniquely and independently determine how to merge those k trees by tree management policy.

Next, each *sponsor* computes $[key, bkey]$ pairs on the *key-path* until either this computation reaches the root or the *sponsor* can not compute a new intermediate key. The *sponsor* broadcast his view of the tree to the group. All members then update their tree views with the new information. If the broadcasting *sponsor* computed the root key, upon receiving the broadcast, all other members can compute the root key as well.

Fig. 5 shows an example of merging two groups, where the *sponsors* M_5 and M_{14} broadcast their trees (T_5 and T_{14}) containing all the *bkeys*, along with BK_5^* and BK_{14}^*. Upon receiving these broadcast messages, every member checks whether it belongs to the *sponsor* in the second round. Every member in both groups merges two trees, and then the *sponsor*(M_5) in this example updates the key tree and computes and broadcasts *bkeys*.

Table 5. Merge Protocol

Step 1 : All *sponsors* M_{s_i} in each T_{s_i}
- generate new share and compute all $[key, bkey]$ pairs on the *key-path* of T_{s_i},
- broadcast updated tree $\widehat{T}_{s_i}$ including only *bkeys*.

$$M_{s_i} \xrightarrow{\widehat{T}_{s_i}(BK^*_{s_i})} \bigcup_{i=1}^{k} C_i$$

Step 2 : Every member
- update key tree by adding new trees and new intermediate nodes,
- remove all *keys* and *bkeys* from leaf node related to the *sponsor* to the root node.

Each *Sponsor* M_{s_t} additionally
- compute all $[key, bkey]$ pairs on the *key-path* until it can proceed,
- and broadcast updated tree $\widehat{T}_{s_t}$ including only *bkeys*.

$$M_{s_t} \xrightarrow{\widehat{T}_{s_t}(BK^*_{s_t})} \bigcup_{i=1}^{k} C_i$$

Step 3 to h (Until a *sponsor* M_{s_j} could compute the group key)
: For each *sponsor* M_{s_t}
- computes all $[key, bkey]$ pairs on the *key-path* until it can proceed,
- and broadcasts updated tree $\widehat{T}_{s_t}$ including only *bkeys*.

$$M_{s_t} \xrightarrow{\widehat{T}_{s_t}(BK^*_{s_t})} \bigcup_{i=1}^{k} C_i$$

Step $h+1$: Every member computes the group key using $\widehat{T}_s$

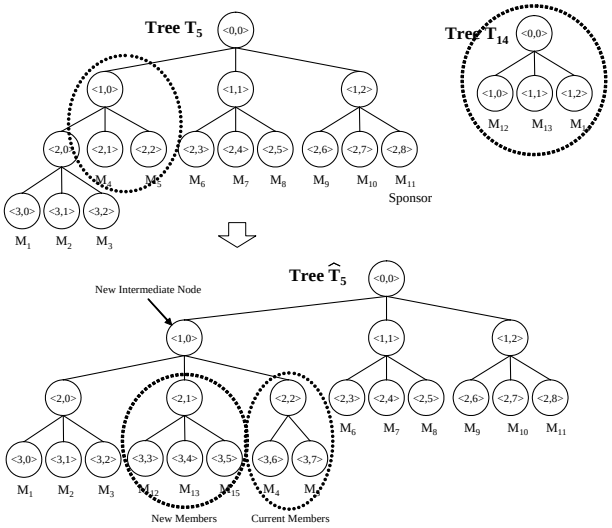

Fig. 5. Tree-updating in merge operation

4 Performance

This section analyzes the communication and computation costs for ***Join***, ***Leave***, ***Merge*** and ***Partition*** protocols. We count the number of rounds, the total number of messages, the serial number of exponentiations, pairings and point multiplications. The serial cost assumes parallelization within each protocol round and presents the greatest cost incurred by any participant in a given round(or protocol). The total cost is simply the sum of all participants' costs in a given round(or protocol).

Table 6 summarizes the communication and computation costs of TGDH and our protocol. The number of current group members, merging groups and leaving members are denoted by n, k and p, respectively. The overhead of protocol depends on the tree height, the balance of the key tree, the location of the joining tree and the leaving nodes. In our analysis, we assume the worst case configuration and list the worst-case cost for TGDH and our protocol.

Since we modified TGDH protocol, the number of communication is equals to TGDH except the number of rounds in merge and key length. But our proposed protocol can reduce the number of computation in each event operation because of low height of key tree. The number of pairings and point multiplications for our protocol depends on whether there exists the subtree with two member nodes or not. We thus compute the cost of average case.

In all events we can reduce the computation cost $O(log_2 n)$ to $O(log_3 n)$. We can get the advantage of the number of computation about 4 times in ***Join***, ***Leave*** and ***Merge*** and 2 times in ***Partition***. The pairings computation is a critical operation in pairings based cryptosystem. The research of pairings imple-

Table 6. Communication and Computation Costs

		Communication		Computation		
		Rounds	Messages	Exponentiations	Pairings	Multiplications
TGDH	Join	2	3	$3\lceil log_2 n \rceil$	0	0
	Leave	1	1	$3\lceil log_2 n \rceil$	0	0
	Merge	$log_2 k + 1$	$2k$	$3\lceil log_2 n \rceil$	0	0
	Partition	$min(log_2 p, h)$	$2\lceil log_2 n \rceil$	$3\lceil log_2 n \rceil$	0	0
Our Protocol	Join	2	3	0	$\lceil log_3 n \rceil - 1$	$\lceil log_3 n \rceil + 1$
	Leave	1	1	0	$\lceil log_3 n \rceil - 1$	$\lceil log_3 n \rceil + 1$
	Merge	$log_3 k + 1$	$2k$	0	$\lceil log_3 n \rceil - 1$	$\lceil log_3 n \rceil + 1$
	Partition	$min(log_3 p, h)$	$2\lceil log_3 n \rceil$	0	$2\lceil log_3 n \rceil$	$2\lceil log_3 n \rceil$

mentation continuously have been studied. Barreto *et al.*[3] proposed an efficient algorithm for pairing-based cryptosystems. In this research we can get the result that computing pairings is about 3 times slower than the modular exponentiation. Therefore our protocol requires less the number of communication and computation than TGDH. However, since involving the pairings computation, our protocol admits of improvement in computational efficiency.

The security analysis of our protocol is in Appendix for details. We describe and prove the Decisional Ternary tree Group Bilinear Diffie-Hellman (DTGBDH) problem.

5 Concluding Remarks

This paper present TGDH group event operation using bilinear map. The modified TGDH using bilinear map support dynamic membership group events with forward and backward secrecy. Our protocol involves pairings operation whose computation is computationally slower than modular exponentiation. However, fast implementation of pairings has been studied actively recently. Since we use ternary key tree, our protocol can use any two-party and three-party key agreement protocol. In this paper, because we use the two-party key agreement protocol using ECDH and the three-party key agreement protocol using bilinear map, the security of our protocol relies on this two protocol. Finally our protocol can reduce the number of computation in group events while preserving the communication and the security property.

References

1. S. Al-Riyami and K. Paterson, "Authenticated three party key agreement protocols from pairings," Cryptology ePrint Archive, Report 2002/035, available at http://eprint.iacr.org/2002/035/.
2. D. Boneh and M. Franklin. "Identity-based encryption from the Weil pairing," Advances in Cryptology-Crypto 2001, LNCS 2139, pp.213-229, Springer-Verlag, 2001. http://www.crypto.stanford.edu/ dabo/abstracts/ibe.html
3. P.S.L.M. Barreto, H.Y. Kim, B.Lynn, and M.Scott, "Efficient Algorithms for pairing-based cryptosystems," To appear in Cryptology-Crypto'2002, available at http://eprint.iacr.org/2002/008/.
4. Wallner, Debby M., Eric J. Harder, and Ryan C. Agee, "Key management for multicast: Issues and architectures," RFC 2627, June 1999.
5. A. Joux, "A one round protocol for tripartite Diffie-Hellman," In W. Bosma, editor, Proceedings of Algorothmic Number Theory Symposium - ANTS IV, volume 1838 of LNCS, pages 385-394. Springer-verlag, 2000
6. A. Joux, "The Weil and Tate Pairings as building blocks for public key cryptosystems," in Algorithm Number Theory, 5th International Symposium ANTS-V, LNCS 2369, Springer-Verlag, 2002, pp. 20-32.
7. N. Koblitz, "Elliptic curve cryptosystems," Mathematics of Computation, vol. 48, pp. 203-209, 1987
8. Y. Kim. A. Perrig and G. Tsudik, "Communication-Efficient Group Key Agreement," IFIP SEC 2001, Jun. 2001.
9. Y. Kim, A. Perrig, G. Tsudik, "Tree-based Group Diffie-Hellman Protocol," ACM-CCS 2000.
10. A. Perrig, D. Song, and J. D. Tyger, "ELK, a New Protocol for Efficient Large Group Key Distribution," In 2001 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 2001.
11. N.P. Smart, "An identity based authenticated key agreement protocol based on the weil pairing," Election. Lett., Vol.38, No.13, pp.630-632, 2002
12. F. Zhang, S. Liu and K. Kim, "ID-Based One Round Authenticated Tripartite Key Agreement Protocol with Pairings," Available at http://eprint.iacr.org, 2002.

Appendix Security Analysis

Here we describe Decisional Ternary tree Group Bilinear Diffie-Hellman (DT-GBDH) problem and apply security proof of TGDH in [9] to the ternary key tree.

For $(q, G_1, G_2, \widehat{e}) \leftarrow g(1^k)$, $n \in N$ and $X = (R_1, R_2, ..., R_n)$ for $R_i \in Z_q^*$ and a key tree T with n leaf nodes which correspond to R_i, we define the following random variables:

- K_j^i : i-th level of j-th key (secret value), each leaf node is associated with a member's session random, *i.e.*, $K_j^0 = R_k$ for some $k \in [1, n]$.
- BK_j^i : i-th level of j-th blinded key (public value), *i.e.*, $K_j^i P$.
- K_j^i is recursively defined as follows:

$$\begin{aligned} K_j^i &= \widehat{e}(P,P)^{K_{3j-2}^{i-1} K_{3j-1}^{i-1} K_{3j}^{i-1}} = \widehat{e}(K_{3j-2}^{i-1} P, K_{3j}^{i-1} P)^{K_{3j-1}^{i-1}} \\ &= \widehat{e}(K_{3j-2}^{i-1} P, K_{3j-1}^{i-1} P)^{K_{3j}^{i-1}} = \widehat{e}(K_{3j-1}^{i-1} P, K_{3j}^{i-1} P)^{K_{3j-2}^{i-1}} \end{aligned}$$

Also we can define public and secret values as below:

- $view(h, X, T) := \{K_j^i P$ where j and i are defined according to $T\}$
- $K(h, X, T) := \widehat{e}(P,P)^{K_1^{h-1} K_2^{h-1} K_3^{h-1}}$

Note that $view(h, X, T)$ is exactly the view of the adversary in our proposed protocol, where the final secret key is $K(h, X, T)$. Let the following two random variables be defined by generating $(q, G_1, G_2, \widehat{e}) \leftarrow g(1^k)$, choosing X randomly from Z_q^* and choosing key tree T randomly from all ternary trees having n leaf nodes:

- $A_h := (view(h, X, T), y)$
- $H_h := (view(h, X, T), K(h, X, T))$

Definition 1. *Let $(q, G_1, G_2, \widehat{e}) \leftarrow g(1^k)$, $n \in N$ and $X = (R_1, R_2, ..., R_n)$ for $R_i \in Z_q^*$ and a key tree T with n leaf nodes which correspond to R_i. A_h and H_h defined as above. **DTGBDH algorithm** $\mathcal{A}_{\mathcal{T}}$ is a probabilistic polynomial time algorithm satisfying, for some fixed $k > 0$ and sufficiently large m:*

$$|Prob[\mathcal{A}_{\mathcal{T}}(A_h) = \text{“}True\text{”}] - Prob[\mathcal{A}_{\mathcal{T}}(H_h) = \text{“}True\text{”}]| > \frac{1}{m^k}$$

*Accordingly, **DTGBDH problem** is to find an Ternary Tree DBDH algorithm.*

Theorem 1. *If the three-party DBDH on G_1, G_2 is hard, then there is no probabilistic polynomial time algorithm which can distinguish A_h from H_h.*

Proof. We first note that A_h and H_h can be rewritten as:
If $X_L = (R_1, R_2, ..., R_l)$, $X_C = (R_{l+1}, R_{l+2}, ..., R_m)$ and $X_R = (R_{m+1}, R_{m+2}, ..., R_n)$ where R_1 through R_l are associated with leaf node in the left tree T_L, $R_l + 1$ through R_m are in the center tree T_C and $R_m + 1$ through R_n are in the right

tree T_R:

$$\begin{aligned}
A_h &:= (view(h, X, T), y) \\
&= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad BK_1^{h-1}, BK_2^{h-1}, BK_3^{h-1}, y) \\
&= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad K_1^{h-1}P, K_2^{h-1}P, K_3^{h-1}P, y) \\
H_h &:= (view(h, X, T), K(h, X, T)) \\
&= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad BK_1^{h-1}, BK_2^{h-1}, BK_3^{h-1}, \widehat{e}(P,P)^{K_1^{h-1}K_2^{h-1}K_3^{h-1}}) \\
&= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad K_1^{h-1}P, K_2^{h-1}P, K_3^{h-1}P, \widehat{e}(P,P)^{K_1^{h-1}K_2^{h-1}K_3^{h-1}})
\end{aligned}$$

We prove this **theorem** by induction and contradiction. The 3-party DBDH problem in G_1 and G_2 is equivalent to distinguish A_1 from H_1. We assume that A_{h-1} and H_{h-1} are indistinguishable in polynomial time as the induction hypothesis. We further assume that there exist a polynomial algorithm that can distinguish A_h from H_h for a random ternary tree. We will show that this algorithm can be used to distinguish A_{h-1} from H_{h-1} or can be used to solve the 3-party DBDH problem.

We consider the following equations:

$$\begin{aligned}
A_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad K_1^{h-1}P, K_2^{h-1}P, K_3^{h-1}P, y) \\
B_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad rP, K_2^{h-1}P, K_3^{h-1}P, y) \\
C_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad rP, r'P, K_3^{h-1}P, y) \\
D_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad rP, r'P, r''P, y) \\
E_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad rP, r'P, r''P, \widehat{e}(P,P)^{rr'r''}) \\
F_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad rP, r'P, K_3^{h-1}P, \widehat{e}(P,P)^{rr'K_3^{h-1}}) \\
G_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad rP, K_2^{h-1}P, K_3^{h-1}P, \widehat{e}(P,P)^{rK_2^{h-1}K_3^{h-1}}) \\
H_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad K_1^{h-1}P, K_2^{h-1}P, K_3^{h-1}P, \widehat{e}(P,P)^{K_1^{h-1}K_2^{h-1}K_3^{h-1}})
\end{aligned}$$

Since we can distinguish A_h and E_h in polynomial time, we can distinguish at least one of (A_h, B_h),(B_h, C_h),(C_h, D_h),(D_h, E_h),(E_h, F_h),(F_h, G_h) or (G_h, H_h).

A_h and B_h: Suppose we can distinguish A_h and B_h in polynomial time. We will show that this distinguisher $\mathcal{A}_{AB_h}$ can be used to solve DTGBDH

problem with height $h-1$. Suppose we want to decide whether $P'_{h-1} = (view(h-1, X_1, T_1), r_1)$ is an instance of DTGBDH problem or r_1 is a random number. To solve this, we generate trees T_2 and T_3 of height $h-1$ with distribution X_2 and X_3, respectively. Note that we know all secret and public information of T_2 and T_3. Using P'_{h-1} and (T_2, X_2), (T_3, X_3) pairs, we generate the distribution:

$$P'_h = (view(h-1, X_1, T_1), view(h-1, X_2, T_2), view(h-1, X_3, T_3), \\ r_1P, K(h-1, X_2, T_2)P, K(h-1, X_3, T_3)P, y)$$

Now we put P'_h as input of $\mathcal{A}_{AB_h}$. If P'_h is an instance of $A_h(B_h)$, then P'_{h-1} is an instance $H_{h-1}(A_{h-1})$.

B_h and C_h: We can generate P'_h by the similar method in (A_h,B_h) and then put P'_h as input of $\mathcal{A}_{BC_h}$ which can distinguish B_h and C_h. If P'_h is an instance of $B_h(C_h)$, then P'_{h-1} is an instance $H_{h-1}(A_{h-1})$.

C_h and D_h: We can generate P'_h by the similar method in (A_h,B_h) and then put P'_h as input of $\mathcal{A}_{CD_h}$ which can distinguish C_h and D_h. If P'_h is an instance of $C_h(D_h)$, then P'_{h-1} is an instance $H_{h-1}(A_{h-1})$.

D_h and E_h: Suppose we can distinguish D_h and E_h in polynomial time. Then, this distinguisher $\mathcal{A}_{DE_h}$ can be used to solve 3-party BDH problem in groups G_1 and G_2. Note that rP, r_1P and r_2P are independent random variable from $view(h-1, X_L, T_L)$, $view(h-1, X_C, T_C)$ and $view(h-1, X_R, T_R)$. Suppose we want to decide whether $(aP, bP, cP, e(P,P)^{abc})$ is a BDH quadruple or not. To solve this, we generate three tree T_1, T_2 and T_3 of height $h-1$ with distribution X_1, X_2 and X_3 respectively. Now we generate new distribution:

$$P'_h = (view(h-1, X_1, T_1), view(h-1, X_2, T_2), view(h-1, X_3, T_3), \\ aP, bP, cP, \widehat{e}(P,P)^{abc})$$

Now we put P'_h as input of $\mathcal{A}_{DE_h}$. If P'_h is an instance of $D_h(E_h)$, then $(aP, bP, cP, \widehat{e}(P,P)^{abc})$ is an invalid(valid) BDH quadruple.

E_h and F_h: We can generate P'_h by the similar method in (A_h,B_h) and then put P'_h as input of $\mathcal{A}_{EF_h}$ which can distinguish E_h and F_h. If P'_h is an instance of $E_h(F_h)$, then P'_{h-1} is an instance $A_{h-1}(H_{h-1})$.

F_h and G_h: We can generate P'_h by the similar method in (A_h,B_h) and then put P'_h as input of $\mathcal{A}_{FG_h}$ which can distinguish F_h and G_h. If P'_h is an instance of $F_h(G_h)$, then P'_{h-1} is an instance $A_{h-1}(H_{h-1})$.

G_h and H_h: We can generate P'_h by the similar method in (A_h,B_h) and then put P'_h as input of $\mathcal{A}_{GH_h}$ which can distinguish G_h and H_h. If P'_h is an instance of $G_h(H_h)$, then P'_{h-1} is an instance $A_{h-1}(H_{h-1})$. □