A Thesis for the Degree of Master

A New Group Key Agreement Scheme based on Ternary Tree

Sang-won Lee

School of Engineering

Information and Communications University

2003

A New Group Key Agreement Scheme based on Ternary Tree

A New Group Key Agreement Scheme based on Ternary Tree

Advisor : Professor Kwangjo Kim

by

Sang-won Lee
School of Engineering
Information and Communications University

A thesis submitted to the faculty of Information and Communications University in partial fulfillment of the requirements for the degree of Master of Science in the School of Engineering

Daejeon, Korea
July. 03. 2003
Approved by

(signed)

Professor Kwangjo Kim
Major Advisor

A New Group Key Agreement Scheme based on Ternary Tree

Sang-won Lee

We certify that this work has passed the scholastic standards required by the Information and Communications University as a thesis for the degree of Master

July. 03. 2003

Approved:

Chairman of the Committee
Kwangjo Kim, Professor
School of Engineering

Committee Member
Myungchul Kim, Associate Professor
School of Engineering

Committee Member
Kyoil Chung, Ph.D
ETRI

M.S. Sang-won Lee
2001807

A New Group Key Agreement Scheme based on Ternary Tree

School of Engineering, 2003, 43p.
Major Advisor : Prof. Kwangjo Kim.
Text in English

Abstract

As group-oriented and collaborative applications is getting popularity, the needs for the secure and reliable group communications increase. One of its important challenges in the secure and reliable group communications is to design secure and efficient group key management.

There are two kinds of group key managements: group key distribution for the centralized management and group key agreement for the distributive management. While group key distribution is often appropriate in large multicast-style groups, not a few collaborative group environments require group key agreement scheme.

Group key agreement scheme is more adaptable than group key distribution in communication environments such as DPG (Dynamic Peer Group) in which its membership can be frequently changed and the communicating party in a group can be dynamically configured.

In this thesis, we focus on the key agreement protocol in DPG environment with low communication and computational cost. As far as the communication and computational cost concerned, the existing group key management protocols such as STR (Steer) and TGDH (Tree-based Group Key Agreement) reduce the cost $O(n)$ to $O(log_2 n)$ by using bi-

nary tree. In the tree-based mechanism, it is noticed that the computational cost can be reduced by lowering the depth of the tree. In this thesis, we propose more efficient key agreement protocols, i.e., skinny tree-based group key agreement and ternary tree-based group key agreement, by applying the pairing-based cryptography to STR and TGDH respectively. Through our proposed protocols, we substitute the binary key tree with the ternary tree and lower the depth of key tree, thus we reduce the computational cost required in protocols.

Unlike most of the existing group key agreement protocols which focus on reducing the computational cost, STR is optimized in communication cost at the expense of the computational cost $O(n)$. However, our skinny tree-based group key agreement reduces the computational cost by the aid of pairing while preserving the communication cost. Thus, it is more profitable in high-delay wide area network. In addition, our ternary tree-based group key agreement reduces the TGDH's computational cost $O(log_2 n)$ to $O(log_3 n)$.

We present the protocol for group membership events in our proposed scheme: member join and leave, group merge and partition. Also we analyze the security of each our proposed schemes based on the hard problem of DBDH (Decision Bilinear Diffie-Hellman). Finally, we measure the performance of our proposed protocols in comparison with STR and TGDH.

Contents

List of Tables

List of Figures

List of Abbreviations

DH Diffie-Hellman

CDH Computational Diffie-Hellman

BDH Bilinear Diffie-Hellman

DBDH Decisional Bilinear Diffie-Hellman

ECDH Elliptic Curve Diffie-Hellman

GDH Group Diffie-Hellman

TGDH Tree-based Group Diffie-Hellman

DTGBDH Decisional Ternary Tree-based Group Bilinear Diffie-Hellman

DSTGBDH Decisional Skinny Tree-based Group Bilinear Diffie-Hellman

List of 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; $i \in \{1,2, ..., N\}$

r_i M_i's session random

br_i M_i's blinded session random, *i.e.* r_iP

h The height of the key tree

$< l, v >$ v-th node at the l-th level in a tree

$RN_{<l>}$ Right member node at level l in skinny tree

$LN_{<l>}$ Left member node at level l in skinny tree

$KN_{<l>}$ Key node at level l in skinny 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 $< i, j >$

$BT_{<i>}$ Tree of member M_i including all of its blinded keys.

BK_i^* set of M_i's blinded keys

P Public information, a point on an elliptic curve

e the Weil Pairing

$\widehat{e}$ the modified Weil Pairing

H_1 Hash function, $H_1 : G_2 \rightarrow Z_q^*$

H_2 Hash function, $H_2 : G_1 \rightarrow Z_q^*$

I. 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.

1.1 Group Key Management

There are different approaches to group key management in peer group. First, centralized group key distribution is that a single key server generates keys and distributes them to the group. Essentially, a key server maintains long-term shared keys with each group member in order to enable secure two-party communication for the actual key distribution. This approach has a drawback: Key server must be always available to every possible subset of a group in order to support continued operation in the event of network reconfiguration.

Another approach, called decentralized group key distribution, includes dynamically selecting a group member that generates and distributes keys to other group members. This approach is more robust and applicable to many-to-many groups since any partition can continue operation by electing a key server. But a key server must establish long-term pair-wise secure channels with all current group members in order to distribute group keys.

Contributory group key agreement method is that each group member contributes an equal share to the common group key. This method can avoid the problems with centralized trust. Moreover contributory method does not require the establishment of pair-wise secure channels among group members.

1.2 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.

Let's summarize all membership operations as follows:

- ***Member Join*** : A new member is added to the group.
- ***Member Leave*** : A member is removed from the group.
- ***Group Merge*** : A group is merged with the current group.
- ***Group Partition*** : A subset of members are split from the group.
- ***Key Refresh*** : The group key is updated.

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.

1.3 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 $\widehat{e} : G_1 \times G_1 \rightarrow G_2$ be a paring which satisfies the following conditions:

1. Bilinear: We say that a map $\widehat{e} : G_1 \times G_1 \rightarrow G_2$ is *bilinear* if $\widehat{e}(aP, bQ) = \widehat{e}(P, Q)^{ab}$ for all $P, Q \in G_1$ and all $a, b \in Z$.

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 $\widehat{e}(P, Q)$ 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.

1.4 Our Contribution

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. We focus on the group key agreement scheme in DPG environment.

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.

In the last two decades a lot of research has been conducted with the aim of minimizing cryptographic overhead in security protocols. It has been long held as an incontrovertible fact that heavy-weight computation is the greatest burden imposed by security protocols. Network devices and communication lines have become significantly faster and cheaper. However, the communication has become both accessible and affordable which resulted in drastic increase in the demand for network bandwidth. Consequently, the explosion in the number of users and their devices often causes network congestion and outages.

STR protocol proposed by Y. Kim *et al.*[8] is a simple, secure and communication-efficient protocol. However the computational cost is proportional to the number of current members. We proposed a new group key agreement protocol that modifies STR protocol by utilizing pairing-based cryptography. The resulting protocol reduces computational cost of STR protocol while preserving the communication cost. This is the first contribution of the thesis.

Y. Kim *et al.*[9] also 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. We present new group key agreement scheme extending TGDH to the group by utilizing pairing based cryptographic. Since our proposed protocol use the ternary key tree, we can reduce the computation cost $O(log_2 n)$ of TGDH to $O(log_3 n)$. This is our second contribution.

Lastly we prove the security of our proposed scheme and compare the performance of our protocols with TGDH and STR.

1.5 Outline of the thesis

In this thesis, we deal with new group key agreement schemes in DPG by utilizing pairing based cryptographic and the security analysis.

The rest of this thesis is organized as follows. In Chapter 2 we briefly describe the basic idea of TGDH and STR. We explains the proposed protocols in Chapter 4. For two kinds of tree we describe protocols of membership events detail: Join, Leave, Merge and Partition. Security and performance analysis is described in Chapter 5. we make a comparison with our proposed protocols, TGDH and STR. We end with conclusion in Chapter 5.

II. Related work

2.1 TGDH

TGDH is an adaptation of key tree in the context of fully distributed, contributory group key agreement. TGDH computes a group key derived from the contribution of all group members using a binary tree.

The tree is organized in the following manner: each node $< l, v >$ is associated with a key $K_{<l,v>}$ and the corresponding blinded key $BK_{<l,v>} = g^{K_{<l,v>}} \bmod p$. The key at the root node is the group key shared by all members, and a key at the leaf node is the random session contribution by a group member. Each member knows all the keys on the path from its leaf node to the root as well as blinded keys on the key tree.

The basic idea here is that every member can compute a group key when all blinded keys on the key tree when all blinded keys on the key tree are known. After any group membership event, every member unambiguously adds or removes some nodes related with the event, and

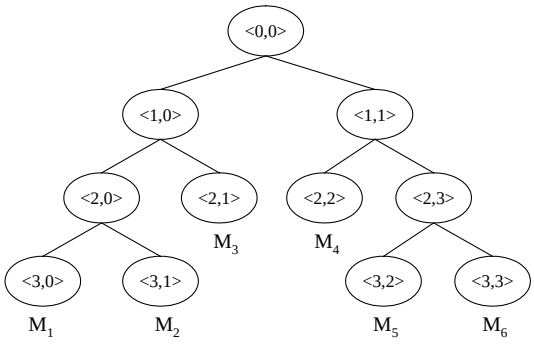

Figure 2.1: A key tree of TGDH

invalidates all keys and blinded keys related with the affected nodes. A special group member, the *sponsor*, then takes on a role to compute keys and blinded keys and to broadcast the key tree to the group. If a sponsor could not compute the group key, then the next sponsor will compute comes into play. Eventually, some sponsor will compute the group key and all blinded keys, and broadcast the entire key tree to facilitate the computation of the group key by the other members of the group.

2.2 STR

STR is basically an "extreme" version of TGDH, where the key tree structure is completely imbalanced or stretched out. Like TGDH, the STR protocol uses a tree structure that associated the leaves with individual random session contributions of the group members. Every internal node has an associated secret key and a public blinded key. The secret key is the result of a Diffie-Hellman key agreement between the node's two children. The group key is the key associated with the root node.

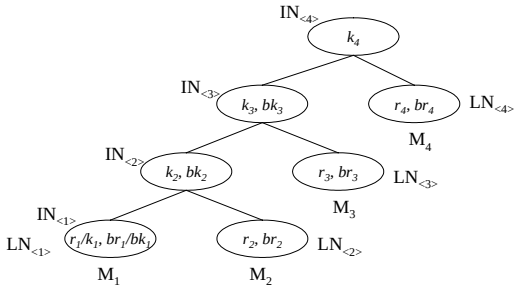

Figure 2.2: A key tree of STR

III. Our Protocol

In this chapter, we present the protocols for group membership changes in proposed group key agreement: ***Join***, ***Leave***, ***Merge*** and ***Partition***.

The proposed protocols use the skinny and ternary tree respectively. 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.

3.1 Skinny Tree-based Group Key Agreement

3.1.1 Notation

Key node $KN_{<l>}$ has three child node: another(lower) key node KN_{l-1} and two member node $RN_{<l-1>}$, $LN_{<l-1>}$. Each leaf node is associated with a specific group member. The exception is $KN_{<1>}$ which is also a leaf node corresponding to M_1.

Each member node $RN_{<i>}(LN_{<i>})$ has a session random r_i chosen and kept secret by M_i. The blinded information of this session random is $br_i(= r_iP)$ called blinded key. Every key node has an associate secret key k_j and blinded key $bk_j(= k_jP)$. The secret key $k_j(j > 1)$ is the result of hash value of pairing operation among the node's three children.

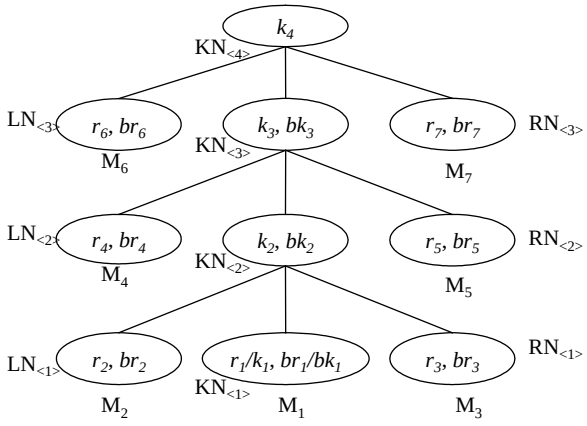

Figure 3.1: An example of a skinny key tree

The shared group key in Fig. 3.1 is the key associated with root node:

$$k_4 = H_1(\widehat{e}(P,P)^{r_6 r_7 H_1(\widehat{e}(P,P)^{r_4 r_5 H_1(\widehat{e}(P,P)^{r_1 r_2 r_3})})})$$

The basic key agreement protocol is as follows: We assume that all members know the structure of the key tree and their initial position with in the tree. Furthermore, each member knows its session random and the blinded session random of all other members. The three member M_1, M_2 and M_3 can first compute the group key corresponding to $KN_{<2>}$. M_1 can computes:

$$\begin{aligned} k_1 &= r_1 \\ k_2 &= H_1(\widehat{e}(r_2 P, r_3 P)^{r_1}) = H_1(\widehat{e}(P,P)^{r_1 r_2 r_3}) \\ k_3 &= H_1(\widehat{e}(br_4, br_5)^{k_2}) = H_1(\widehat{e}(P,P)^{k_2 r_4 r_5}) \\ &... \\ k_h &= H_1(\widehat{e}(P,P)^{k_{n-1} r_{2n-2} r_{2n-1}}) \end{aligned}$$

Next, M_1 broadcasts $BT_{<1>}$ with all blinded key bk_i $(1 \leq i \leq \frac{N-1}{2})$ and $br_j (1 \leq j \leq N)$. Upon receiving the broadcasted message, every member can compute the group key k_h.

In top level, the root node has two or three child nodes due to the number of members: unbalanced or balanced key tree. In unbalanced key tree we can not use pairing. Thus we compute the group key using EC-DH(Elliptic Curve Diffie-Hellman).

3.1.2 Join Protocol

We assume the group has n members ($\{M_1, M_2, ..., M_n\}$). When the new member M_{n+1} join the group, both the new member and the current group member receive this notification simultaneously. The new member M_{n+1} broadcasts a join request message that contains its own blinded key bk_{n+1}.

At this time, we are faced with two types of the current tree. In the case of the unbalanced tree, each member M_i can compute the group key by inserting the new member into the blank node. The *sponsor* broadcast the blinded key tree $BT_{<n>}$ to M_{n+1} without computing bk_n. However in balanced tree, each member M_i promote the new root node $KN_{<n/2+1>}$ and add the new member. Then *sponsor* M_n updates his own session random, computes br_n, k_n, bk_n and broadcasts $BT_{<n>}$ to the group with all blinded keys and blinded session randoms.

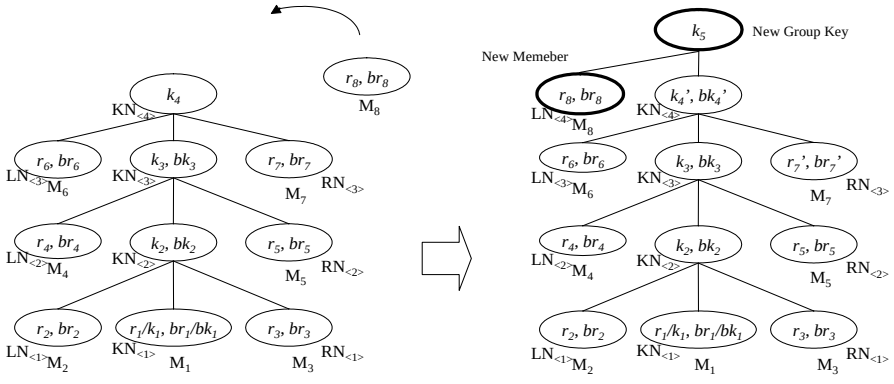

Figure 3.2: Join operation in the skinny tree

All existing members only need the new member's blinded session random and the new member needs the blinded group key(bk_n) of the prior group and the blinded session random(br_n) of the sponsor. Therefore all member can compute the group key. In ***Join***, the *sponsor* is always topmost leaf node, *i.e.*, the most recent member in the current group.

Fig. 3.2 shows the example of the new member joining the group. The *sponsor* M_7 update his own session random r_7 and then compute br_7, k_4, bk_4. The sponsor broadcast $BT_{<7>}$ to the group with all blinded session randoms and blinded keys. Upon receiving the message, all members can compute the new group key($k_5 = H_2(r_8 k_4 P)$).

3.1.3 Leave Protocol

We assume that a member M_d leaves the group of n members. In ***Leave***, the *sponsor* is the top rightmost leaf node. If M_d leaves the group, each remaining member updates his key tree by deleting the member nodes corresponding to M_d and its parent node. The sponsor moves his position in key tree to the blank node M_d of the leaving member, updates his own session random and computes all keys and blinded keys up to the root, and then broadcast $BT_{<n>}$. This information allows all members to recompute the new group key.

In Fig. 3.3, the member M_5 leaves the group. the *sponsor* M_7 move his position to the leaving point. Then M_7 updates his own random session r'_7, computes br'_7, k'_3, bk'_3, and broadcast $BT_{<7'>}$ to the all members. Upon receiving the broadcast message, all members can compute the new group key. Though the leaving member M_5 knows all blinded keys, he can not compute the new group key since his random session is not the part of the group key.

The ***Leave*** protocol provides forward secrecy since a former member

cannot compute the new key owing to the *sponsor*'s changing the session random. Due to the *sponsor* refreshing his own random, the protocol also supports key independence.

3.1.4 Merge Protocol

We assume that, as in the case of ***Join***, the group member can receive the merge event simultaneously. It is natural to merge the smaller group onto the higher one.

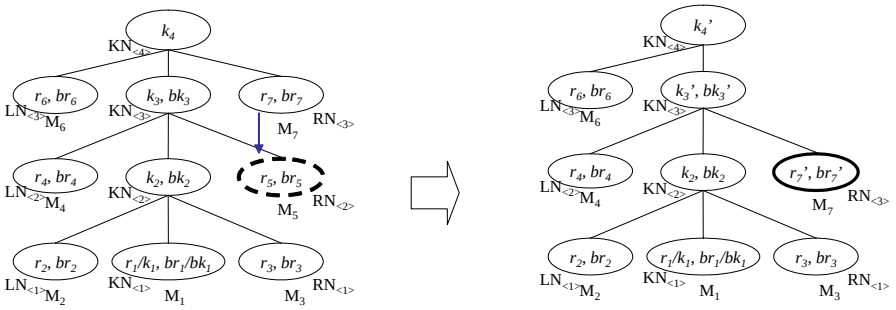

Figure 3.3: Leave operation in the skinny tree

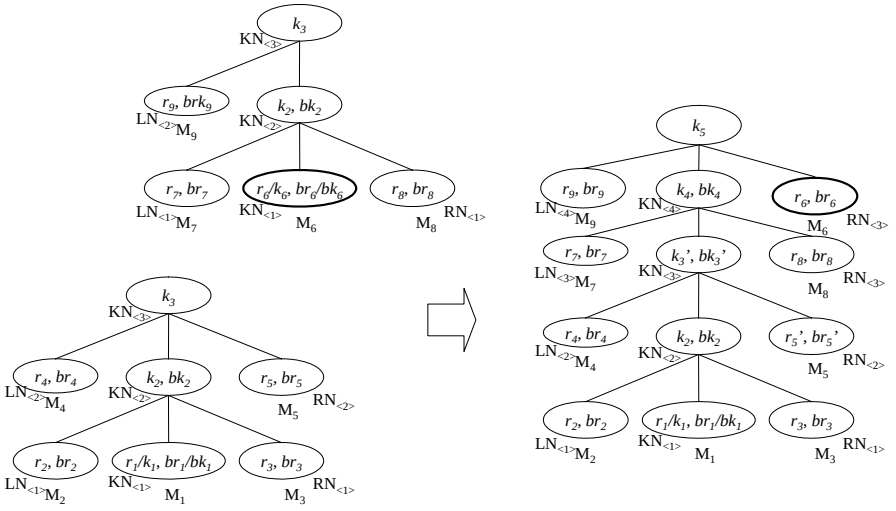

Figure 3.4: Merge operation in the skinny tree

If the lower tree is the unbalanced tree, the *sponsor* of the upper tree moves M_1, M_2 to the blank upper node. If there is not the blank node, the *sponsor* generates new root node and the blank leaf node, and then moves them to the blank node. Finally trees are merged by inserting M_3 of the lower tree to the blank node of upper tree. We can merge trees by using $M_2(, M_3)$ and key node of lower tree. Also we can merge many trees by using this method recursively.

In Fig. 3.4, in the first round of the ***Merge*** protocol, each sponsor M_5, M_9 broadcasts his key tree with all blinded key and blinded session random. Upon receiving this message, all member in two groups can updates key tree. Each member moves M_6 to the upper blank node and insert key node $KN_{<3>}$ of lower tree. Note that the upper nodes should be renumbered. In second round. the *sponsor* M_5 computes all keys and blinded keys up to the root node, and broadcasts $BT_{<5>}$ with all blinded keys and blinded session randoms. All member now have the complete set of blinded keys, which allows them to compute the new group key.

3.1.5 Partition Protocol

The partition of the group is caused by a network fault. To the remaining members, this actually appears as a concurrent Leave of multiple members. We can handle multiple leaving members in a single round by modifying the ***Leave*** protocol.

After deleting all leaving nodes, the *sponsor* M_s fills the topmost members into the leaving nodes and refreshes his session random, computes keys and blinded keys going up the tree. Then the sponsor broadcasts $BT_{<s>}$ containing only blinded keys. Each member including M_s can computes the group key.

3.2 Ternary Tree-based Group Key Agreement

3.2.1 Notation

Fig. 3.5 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. 3.5, 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. 3.5, 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 :

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

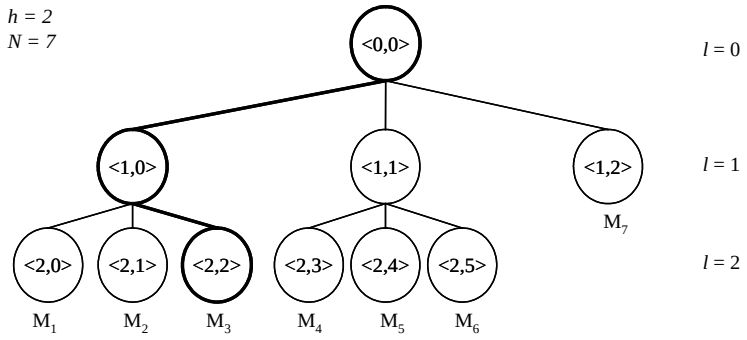

Figure 3.5: An example of a ternary key tree

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.

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.

3.2.2 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

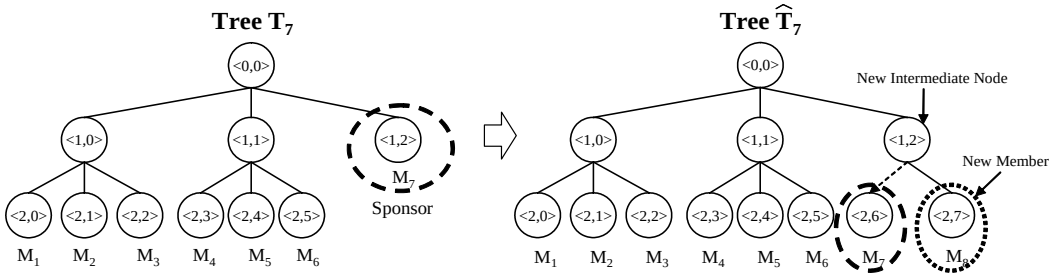

Figure 3.6: Join operation in the ternary tree

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.

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 $< 2, 7 >$.

Table 3.1: 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 $\widehat{T}_s$.

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.

3.2.3 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

Table 3.2: 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 $\widehat{T}_s$.

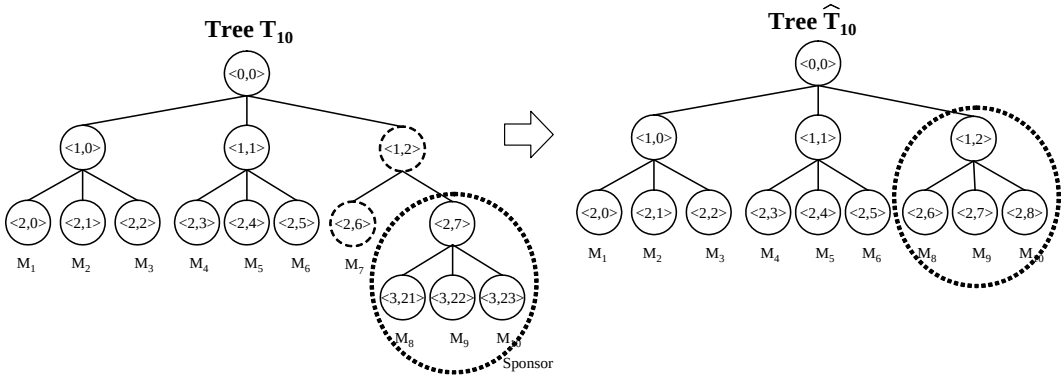

Figure 3.7: Leave operation in the ternary tree

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. 3.7, 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.

In Fig. 3.7, 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.

3.2.4 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***

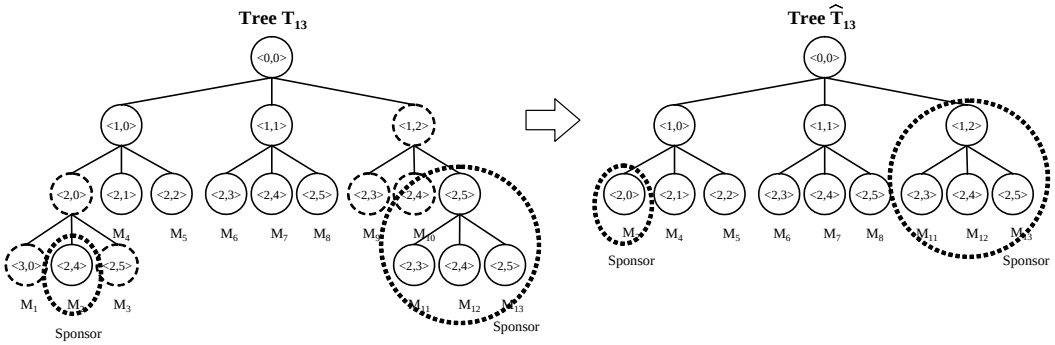

Figure 3.8: Partition operation in the ternary tree

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 3.3.

Fig. 3.8 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 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.

Table 3.3: 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 $\widehat{T}_s$.

3.2.5 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 right-most 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. 3.9 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 3.4: 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 $\widehat{T}_s$

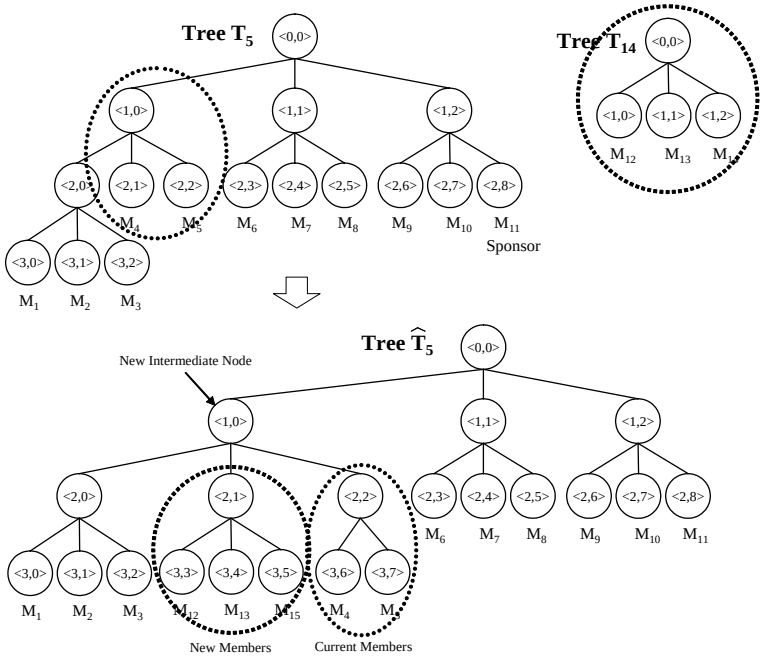

Figure 3.9: Merge operation in the ternary tree

IV. Analysis

We prove the security of the skinny and ternary tree-based group key agreement protocol. We assume the passive adversary who can obtain the all of the blinded secrets and the blinded keys. The security of our protocols is based on the hardness of DBDH(Decision Bilinear Diffie-Hellman) problem. Also we measure the performance of our proposed protocols, STR and TGDH. The results show the advantage of reduced computational costs.

4.1 Security Analysis

4.1.1 Decisional Skinny tree Group Bilinear Diffie-Hellman Problem

Fig. 4.1 shows the structure and the notation for the balanced skinny 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

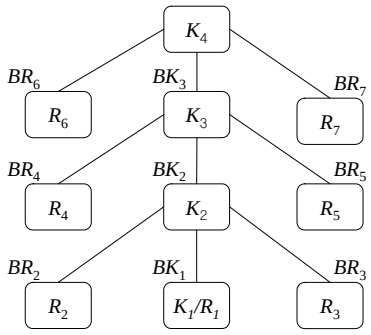

Figure 4.1: Notation for ballanced skinny tree

$R_i \in Z_q^*$ and a skinny key tree ST with n leaf nodes which correspond to R_i, we define the following random variables:

- K_i : i-th level key (secret value)
- BK_i : i-th level blinded key (public value), *i.e.*, K_iP.
- R_j : i-th node's session random chosen uniformly $\in_R G$.
- BR_j : i-th node's blinded session random, *i.e.*, R_iP.
- K_i is recursively defined as follows:

$$\begin{aligned} K_j^i &= \widehat{e}(P,P)^{K_{i-1}R_{2i-1}R_{2i-2}} = \widehat{e}(BR_{2i-1}, BR_{2i-2})^{K_{i-1}} \\ &= \widehat{e}(BK_{i-1}, BR_{2i-1})^{R_{2i-2}} = \widehat{e}(BK_{i-1}, BR_{2i-1})^{R^{2i-1}} \end{aligned}$$

  K_i and R_j are secret, and BK_i and BR_j are public.
  We also define the following random variables:

- $view(h, X, ST) := \{K_iPandR_jP$ where j and i are defined according to $ST\}$
- $K(h, X, ST) := \widehat{e}(P,P)^{R_{2h-1}R_{2h-2}K_h}$

Note that $view(h, X, ST)$ is exactly the view of the adversary in our proposed protocol, where the final secret key is $K(h, X, ST)$. 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 a skinny key tree ST randomly from all ternary trees having n leaf nodes:

- $A_h := (view(h, X, ST), y)$
- $D_h := (view(h, X, ST), K(h, X, ST))$

Definition IV.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 skinny key tree ST with n leaf nodes which correspond to* R_i. A_h *and* D_h *defined as above.* ***DSTGBDH algorithm*** $\mathcal{A}_{\mathcal{ST}}$ *is a probabilistic polynomial time algorithm satisfying, for some fixed* $k > 0$ *and sufficiently large m:*

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

Accordingly, ***DSTGBDH problem*** *is to find an Skinny Tree DBDH algorithm.*

Theorem IV.2 *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* D_h.

Proof: Assume that there exists a polynomial algorithm that can distinguish between A_h and D_h. We will show that this algorithm can be used to distinguish A_{h-1} and D_{h-1} or solve the 3-party BDH problem.

Consider the following equations when $X_1 = (R_1, R_2, ..., R_{n-1})$ and ST_1 is a subtree rooted the center child of the root node.

$$\begin{aligned}
A_h &:= (view(h-1, X, ST), BK_{h-1}, BR_{2h-1}, BR_{2h-2}, y) \\
B_h &:= (view(h-1, X, ST), r'p, BR_{2h-1}, BR_{2h-2}, y) \\
C_h &:= (view(h-1, X, ST), r'P, BR_{2h-1}, BR_{2h-2}, \widehat{e}(P,P)^{r'R_{2h-1}R_{2h-2}}) \\
D_h &:= (view(h-1, X, ST), K(h-1, X, ST)P, \\
&\quad BR_{2h-1}, BR_{2h-2}, \widehat{e}(P,P)^{K(h-1,X,ST)R_{2h-1}R_{2h-2}})
\end{aligned}$$

Since we can distinguish A_h and D_h in polynomial time, we can distinguish at least one of (A_h, B_h),(B_h, C_h) or (C_h, D_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 DSTGBDH problem with height $h-1$. Suppose we want to decide whether $P'_{h-1} = (view(h-1, X', ST'), r_1)$ is an instance of

DSTGBDH problem or r_1 is a random number. To solve this, we generate random number r_2 and r_3 and compute r_2P and r_3P. Using P'_{h-1} and (r_2, r_3, r_2P, r_3P), we can generate the distribution:

$$P'_h \quad = \quad (view(h-1, X', T'), r_1P, r_2P, r_3P, y)$$

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 $D_{h-1}(A_{h-1})$.

B_h and C_h: Suppose we can distinguish B_h and C_h in polynomial time. Note that $r'P$ is an independent variable from $view(h-1, X, ST)$. Suppose we want to test whether $(aP, bP, cP, \widehat{e}(P,P)^{abc})$ is a BDH quadruple or not.

To solve this, we generate a key tree ST' of height $h-1$ with distribution X'. Now we generate a new distribution:

$$P'_h \quad = \quad (view(h-1, X', T'), aP, bP, cP, \widehat{e}(P,P)^{abc})$$

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

C_h and D_h: Suppose we can distinguish C_h and D_h in polynomial time. We will show that this distinguisher $\mathcal{A}_{CD_h}$ can be used to solve DSTGBDH problem with height $h-1$. Suppose we want to decide whether $P'_{h-1} = (view(h-1, X', ST'), r_1)$ is an instance of DSTGBDH problem or r_1 is a random number. To solve this, we generate random number r_2 and r_3 and compute r_2P and r_3P. Using P'_{h-1} and (r_2, r_3, r_2P, r_3P), we can generate the distribution:

$$P'_h \quad = \quad (view(h-1, X', T'), r_1P, r_2P, r_3P, \widehat{e}(P,P)^{r_1r_2r_3})$$

Now we put P'_h as input of $\mathcal{A}_{CD_h}$. If P'_h is an instance of $C_h(D_h)$, then P'_{h-1} is an instance $D_{h-1}(A_{h-1})$.

4.1.2 Decisional Ternary tree Group Bilinear Diffie-Hellman Problem

In this section we describe Decisional Ternary tree Group Bilinear Diffie-Hellman (DTGBDH) problem and apply security proof of TGDH in [9] to the ternary key tree. Fig. 4.2 is an example of a key tree when $n = 9$.

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:

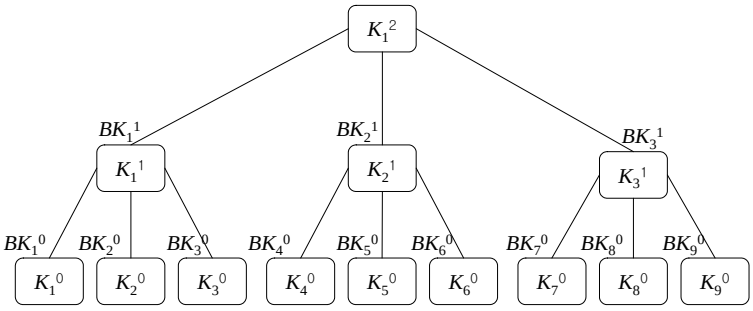

Figure 4.2: Notation for fully ballanced ternary tree

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

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 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 IV.3 *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 IV.4 *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:

$$\begin{aligned}
A_h &= (view(h-1, X_L, T_L), view(h-1, X_C, T_C), view(h-1, X_R, T_R), \\
&\quad K_1^{h-1}P, K_2^{h-1}P, K_3^{h-1}P, y)
\end{aligned}$$

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

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:

$$
\begin{aligned}
P'_h &= (view(h-1, X_1, T_1), view(h-1, X_2, T_2), view(h-1, X_3, T_3), \\
&\quad r_1P, K(h-1, X_2, T_2)P, K(h-1, X_3, T_3)P, y)
\end{aligned}
$$

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:

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

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 $A_{h-1}(H_{h-1})$.

4.2 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 4.1 summarizes the communication and computation costs for the four protocols. The number of current group members, merging groups and leaving memebers 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 the four protocols.

Table 4.1: Communication and Computation Costs

		Rounds	Messages	Exponentiations	Pairings	Multiplications
STR	Join	2	2	4	0	0
	Leave	1	1	$\frac{3}{2}n+2$	0	0
	Merge	2	$k+1$	$3m+1$	0	0
	Partition	1	1	$\frac{3}{2}n+2$	0	0
Our Skinny Protocol	Join	2	2	0	1	$\frac{3}{2}$
	Leave	1	1	0	$\frac{n}{4}$	$\frac{3}{10}n$
	Merge	2	$k+1$	0	$\frac{m+1}{2}$	$\frac{m+3}{2}$
	Partition	1	1	0	$\frac{n}{4}$	$\frac{3}{10}n$
TGDH	Join	2	3	$\frac{3}{2}\lceil log_2 n\rceil$	0	0
	Leave	1	1	$\frac{3}{2}\lceil log_2 n\rceil$	0	0
	Merge	$log_2 k+1$	$2k$	$\frac{3}{2}\lceil log_2 n\rceil$	0	0
	Partition	$min(log_2 p, h)$	$2\lceil log_2 n\rceil$	$3\lceil log_2 n\rceil$	0	0
Our Ternary 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	$2\lceil log_3 n\rceil$	$2\lceil log_3 n\rceil$

As seen from the table, STR and our skinny tree is minimal in communication on every membership event. 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 the case of the skinny tree, we can get the advantage of the number of computation about 6 times in all event except ***Join***. This result is caused by the lowered height of the key tree. In all events of the ternary tree 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 implementation 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.

V. Conclusion and further work

In secure and reliable communication, to manage the conference key is critical problem. Several key management methods have been proposed for secure communication. However we focus on the group key agreement scheme in DPG environment.

This thesis propose new group key agreement protocol and present dynamic group event operation in the skinny tree and ternary tree using bilinear map. We modify STR and TGDH protocol by utilizing pairing-based cryptography. Our protocols support dynamic membership group events with forward and backward secrecy. Our protocols involve 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.

As the future work, it remains as an open problem to efficiently implement and integrate the proposed protocols. The other problem is to support the additional functions such as authentication in lower complexity of communication and computation.

삼진 트리에 기반한 새로운 그룹키 합의 방식

이상원

안전하고 신뢰할 수 있는 그룹 통신은 최근 그룹 및 그룹 구성원간의 협조가 필요한 응용 분야가 발전하면서 점차 그 필요성이 대두되고 있다. 안전하고 믿을 수 있는 통신을 위한 중요한 문제 중 하나가 안전하고 효율적인 그룹키 관리이다.

멤버십이 쉽게 바뀌고 통신 상대가 동적으로 설정될 수 있는 애드혹 그룹에 속하는 DPG(Dynamic Peer Group)에서의 그룹키 관리에 관심을 갖는다. 그룹키 관리방법에는 다음과 같은 두 가지가 있다: 그룹키 분배방식과 그룹키 합의방식. 중앙집중적인 관리는 거대한 멀티캐스트 형태의 그룹에서 키 배포에 적합한 반면, 많은 협동적인 그룹 설정은 분산된 키 합의를 요구한다. 그룹키 합의 개념은 그룹키 분배보다 네트워크가 빈번하게 분할되는 DPG와 같은 통신 환경에 더 적합하다. 결국 DPG 환경에서 그룹키 관리방법으로써 그룹키 합의 프로토콜에 중점을 둔다. 통신량과 계산량은 DPG에서 그룹키 합의 개념에서 중요한 사항이다. 많은 그룹키 관리 프로토콜이 이진 트리를 사용함으로써 계산량을 $O(n)$에서 $O(log_2 n)$으로 줄일 수 있기 때문에, 많은 그룹키 관리 프로토콜들은 그룹키의 효율적인 계산을 위해 이진 트리를 채용한다.

본 논문에서는 새로운 그룹키 합의 개념을 제시한다: Skinny 트리 기반 그룹키 합의와 삼진(Ternary) 트리 기반 그룹키 합의. 이 개념들은 STR(Steer)와 TGDH(Tree-based Group Key Agreement)에 기초하며 STR과 TGDH에 Skinny 트리와 삼진 트리를 사용함으로써 페어링 기반의 암호기술을 적용한다. 이진 트리를 삼진 트리로 대체하고 키 트리의 높이를 낮춘다. 결국 프로토콜에서 요구하는 계산량이 줄어들

게 된다.

대부분의 이전 그룹키 합의 프로토콜들은 프로토콜에서 요구하는 계산량을 줄이는데 중점을 두었다. 하지만 완전 비대칭 이진 트리를 사용하는 STR은 $O(n)$의 계산량을 요구하는 반면, 통신량에 있어서 최적화 되어 있다. 따라서 제안 프로토콜은 통신량을 유지한 채 Skinny 트리를 사용함으로써 계산량을 줄일 수 있다. 제안 프로토콜은 통신에 있어 최적화되어 있으므로 고-지연 광대역 네트워크에 적합하다.

또한 계산 효율성을 개선하기 위해 디피-헬만 키 교환을 이진 트리에 활용한 TGDH에 페어링 기반의 암호기술을 적용한 새로운 그룹키 합의 프로토콜을 제안한다. 이 프로토콜에서 우리는 TGDH의 계산량을 줄이기 위해 삼진 트리를 사용하고 계산량은 $O(log_2 n)$에서 $O(log_3 n)$으로 줄어든다.

제안된 프로토콜에서 그룹 멤버십 이벤트에 대한 프로토콜을 제시한다: 구성원의 가입과 탈퇴, 그룹의 합병과 분할. 또한 Skinny 트리와 삼진 트리에 기반한 제안 프로토콜들의 안전성에 대하여 분석한다. 제안 프로토콜의 안전성은 DBDH의 문제의 어려움에 기반한다. 마지막으로 제안 프로토콜을 STR, TGDH와 비교함으로써 성능에 대해 측정한다.

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," ACMCCS 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.

Acknowledgement

First, I would like to express my sincere gratitude to Prof. Kwangjo Kim, my academic advisor, for his constant direction and support. He always has shown his consistent affection and encouragement for me to carry out my research and life in ICU. Special thanks also goes to Prof. Myungchul Kim and Dr. Kyoil Chung for their generosity and agreeing to serve as committee members of my thesis.

I also would like to thanks to all members of cryptology and information security laboratory: Zeen Kim, Hyungki Choi, Kyusuk Han, Byunggon Kim, Songwon Lee, Hwasun Chang, Chuljoon Choi, Sungjoon Min, Joongman Kim, Jaehyrk Park, Soogil Choi, Jeongkyu Yang, Seokkyu Kang, Vo Duc Lim and Dang Nguyen Duc from Vietnam, Yan Xie, Xiaofeng Chen and Ping Wang from Chaina, and Divyan from India, for giving me lots of interests and good advices during the course of my study. In addition, I appreciate to the graduates, Myungsun Kim, Jungin Kim and Jungyeon Lee, for their everlasting guidance in life and study of ICU.

Most of all, I should mention my father and mother for their endless concerns and devotional affection. I cannot forget their trust and encouragement on me. My brothers also have given me warmhearted concerns. I hope God bless my family and to be happy.

Finally, I will always remember the life of ICU. It filled up my poor knowledge and made me a grown-up person.

Curriculum Vitae

Name : Sang-won Lee

Date of Birth : May. 25. 1975

Sex : Male

Nationality : Korean

Education

1993.3–2000.2	Engineering Chungnam University (B.S.)
2001.6–2003.6	Engineering Information and Communications University (M.S.)

Career

2001.6–2001.11	Graduate Research Assistant A Study on the Algorithms for Counting Points of an Elliptic Curve NSRI
2001.6–	Graduate Research Assistant A Study on the Core Information Security Technology Information Research center for Information Security

2001.6–2001.11 Graduate Research Assistant
A Study on Cryptanalysis of the Block Cipher Candidates for AES, NESSIE and CRYPTREC
Korea Information Security Agency

2002.4–2002.11 Graduate Research Assistant
Development of secure electronic trading system for online game items
CEMTLO Media.

2002.6–2002.11 Graduate Research Assistant
Research on ID-Based PKI and its Applications
Electronics and Telecommunications Research Institute

Academic Experience

2003.2– KIISC student member

2002.12 제5회 정보보호 우수논문 공모 장려상 수상, 한국정보보호진흥원

Publications

(1) 2003.7 이상원, 김진, 김광조, "트리 기반 그룹키 인증 및 합의 프로토콜", CISC2003 한국정보보호학회 하계 종합학술 발표회

(2) 2003.7 김진, 이상원, 최철준, 김광조, "IPsec용 차세대 키관리 프로토콜의 동향 및 분석", CISC2003 한국정보보호학회 하계 종합학술 발표회

(3) 2003.6 이상원, 천정희, 김용대, "Pairing을 이용한 트리 기반 그룹키 합의 프로토콜", 한국정보보호학회 논문지 게제 예정

(4) 2003.5 Sang-won Lee, Kwangjo Kim, Youngdae Kim, Dae-Hyun Ryu, "An Efficient Tree-based Group Key Agreement using Bilinear Map", To appear in ACNS'03, Kunming, China.

(5) 2002.12 이상원, "Pairing을 이용한 트리 기반 그룹키 합의 프로토콜", 제 5회 정보보호 우수논문 공모 장려상, 한국정보보호진흥원