Provably Authenticated Group Key Agreement based on Braid Groups – The Dynamic Case

Pipat Hiranvanichakorn
Graduate School of Applied Statistics, National Institute of Development Administration Bangkok
118, Seri Thai Road, Klong-Chan, Bangkapi, Bangkok 10240, Thailand
(Email: pipat@as.nida.ac.th)

(Received Jan. 28, 2016; revised and accepted Apr. 23 & July 31, 2016)

Abstract

Most group key agreement protocols make use of modular exponential operations which require extensive computing resources in devices. Thus, they are unsuitable for resource- constrained devices such as mobile phones, smart cards and intelligent watches. This paper proposes a group key agreement protocol based on braid groups which requires only multiplication operations. The proposed protocol is a scalable one and needs only two rounds for setting a secure group communication. To prevent a man-in-the-middle attack, exchanged messages are simply authenticated by using users' long-term public and private keys instead of signatures. Authentication proofs are also made by using well-known BAN logic. The protocol is designed for dynamic group communication in which member join, member leave, group merge and group partition are discussed. The proposed algorithms take advantage of precomputed values achieved in previous sessions to update keys in subsequent session. This makes the scheme generates fewer communicating messages and lessens user devices' computation. Comparisons of security and complexity among several two-round protocols are also discussed in this article.

Keywords: Authenticated group key agreement, BAN authentication logic, braid groups

1 Introduction

Group communication has been widely studied in recent times because it has many commercial applications. These include, amongst others, audio-video conference, pay-per-view, audio-video broadcasting, stock quote services, collaborate tasks, and so on. Many of these applications require security services such as, data confidentiality, data integrity and data authentication, during transmission. Group key management can be used for generating group session key in order to provide secure communication. There are two main types of group key management protocols: group key distribution and group key agreement. In group key distribution scheme, the group session key is generated and distributed by a central trusted party via secure channel. The main disadvantage of this scheme is that it needs both a trust authority as well as the availability of secure channels. Furthermore, the scheme suffers from security risks when there is a single point of failure. A group key agreement scheme, by contrast, allows all authorized group members to work together to establish a group session key. These schemes [1, 3, 5, 6, 9, 12, 15, 16, 18, 21] have been studied widely in recent years because they are considered to be scalable.

Most group key agreement protocols are based on Diffie-Hellman as well as Elliptic Curve two-party key exchange protocol and they require exponential operations [3, 7, 8, 9, 12, 14, 15, 16, 18]. However, there are studies based on braid groups which require only multiplication operations [1, 6, 10, 11, 13, 19, 20]. For instance, Lee et al. [13] has proposed an authenticated group key agreement protocol for designated groups based on braid groups. In authenticated group key protocol, each member is assured that no users outside the group can find out the session key. The key disadvantage of this work is that the number of rounds needed in the work is linear to the number of group members. In each round, some calculated values have to be sent from user i to user $i+1$ for further calculation. Therefore the scheme can suffer from long delays at some participants in the group. In a recent paper [1], Aneksrup and Hiranvanichakorn proposed a dynamic group key agreement based on braid groups and a key tree structure. In a dynamic group, parties may join and leave the group any given number of times. This scheme needs $O(n)$ rounds for n-parties group initialization, constant rounds in the case that the user n is the group director for join operation, and $O(n)$ rounds in worst case for leave operation. However, an $O(n)$-round protocol is considered not to be scalable and suffers from long network delays. To get around these problems, some researchers have turned to constant-round protocols. Nevertheless, those works are still based on exponential operation cryptosystems [7, 8, 12, 14, 21].

In the paper by Hwang et al. [8], a framework for extending a two-party key exchange protocol to a contributory group key one has been proposed by using a ring structure of participants. Hwang et al. applied their work to Diffie-Hellman key exchange, resulting in a two-round group key agreement protocol. However, Lee et al. [14] pointed out that Hwang et al.'s scheme has some flaws when applied to a dynamic group. Lee et al. also introduced an improvement to the algorithm to get rid of the flaws. Nonetheless, it can be shown in this paper that Lee et al.'s algorithm as well as several ring-structure protocols still suffer some flaws when they are applied in dynamic environment.

In this paper, an authenticated group key agreement protocol using braid group cryptography is proposed. The proposed protocol needs only two rounds and uses a ring structure of participants. Authentication between communicating users can be simply done by using users' long-term private and public keys instead of digital signature scheme. The protocol is a dynamic case in which member join, member leave, group merge and group partition are discussed. In the protocol, precomputed values in previous sessions are used for updating session keys in subsequent session. This can generate fewer exchanged messages and lessen users' computation than previously proposed protocols. In addition, an authentication proof using BAN authentication logic [4, 17] is given in this article. The proof for the proposed group key agreement is different from the authentication proof given in Lee et al [14]. Security and performance analysis of the proposed protocol are also discussed. Finally, comparisons among ring-structure protocols are illustrated.

The rest of this paper is organized as follows. Section 2 provides some preliminaries of braid group cryptography. A provably authenticated key exchange protocol based on braid groups is described in Section 3. In Section 4, some reviews of previous group key agreement protocols and their security analysis are given. The proposed authenticated group key agreement protocol for dynamic group is described in Section 5. Section 6 offers a security analysis of the proposed protocol showing the authentication proof. In Section 7, security analysis of the protocol against some well-known attacks and some comparisons among ring-structure protocols are provided. The conclusion is given in Section 8.

2 Preliminaries

This section gives a brief description of braid groups, some hard problems in braid groups as well as a well-known key exchange protocol proposed by Ko et al. [11]. For more information on braid groups, please refer to papers [2, 10, 11].

The n-braid group B_n is the group generated by generators $\sigma_1, \cdots, \sigma_{n-1}$ with the relations,

1) $\sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j$ where $|i-j| = 1$, e.g. $\sigma_3\sigma_2\sigma_3 = \sigma_2\sigma_3\sigma_2$;

2) $\sigma_i\sigma_j = \sigma_j\sigma_i$ where $|i-j| \geqslant 2$, e.g. $\sigma_5\sigma_3 = \sigma_3\sigma_5$.

Each element of the group B_n is called an n-braid. There are a number of mathematically hard problems in braid groups, one of the most famous of which is the Generalized Conjugacy Search Problem (GCSP). GCSP states that two braids x and y are conjugate if there exists a braid a such that $y = axa^{-1}$, where a^{-1} is the inverse of a. For $m < n$, B_m which is a subgroup of B_n generated by $\sigma_1, \cdots, \sigma_{m-1}$, the hardness of GCSP is as follows.

- Given $(x, y) \in B_n \times B_n$ such that $y = bxb^{-1}$ for some $b \in B_m$, $m \leq n$.

- The objective is to find $a \in B_m$ such that $y = axa^{-1}$.

It is considered to be the case that, for sufficiently large braids, it is easy to compute y from b and x. However, exponential time is needed to compute a, even when y and x are known.

In paper [11], Ko et al. has also stated that for $a, b \in B_n$, it is hard to guess a or b from ab.

GCSP is applied in several cryptographic protocols [1, 6, 10, 11, 13, 19, 20]. There are also several attempts to solve GCSP in braid groups. In a paper [10], Ko et al. has stated that the attacks on braid cryptosystems were successful because the current ways of random key generation almost always result in weaker instances of the conjugacy problem. They then proposed several ways of generating secure keys for braid cryptography.

2.1 Ko-Lee Key Agreement Protocol

Ko et al. has proposed a well-known key agreement protocol based on GCSP. The protocol includes the following steps.

1) Preparation step: When A(lice) and B(ob) want to establish a shared key, an appropriate pair of integers (l, r) and a sufficiently complicated $(l + r)$ braid $x \in B_{l+r}$ are selected and published as system parameters.

2) Key agreement implementation:

   (a) A chooses a random secret braid $a \in LB_l$, where LB_l is a subgroup of B_{l+r} and generated by $\sigma_1, \cdots, \sigma_{l-1}$. A then sends $y_1 = axa^{-1}$ to B. The braid y_1 is considered to be A's public key.

   (b) B chooses a random secret braid $b \in LB_r$ where LB_r is a subgroup of B_{l+r} and generated by $\sigma_{l+1}, \cdots, \sigma_{l+r-1}$. B then sends $y_2 = bxb^{-1}$ to A. The braid y_2 is also considered to be B's public key. By the commutative property of braid groups, we have $ab = ba$.

   (c) Upon receiving y_2, A computes the shared key, $k_{AB} = ay_2a^{-1} = abxb^{-1}a^{-1}$.

   (d) B can also compute $k_{AB} = by_1b^{-1} = baxa^{-1}b^{-1} = abxb^{-1}a^{-1}$.

In the above scheme, even if E(ve) sees the message axa^{-1} (and bxb^{-1}), she needs exponential time to compute a (and b). Therefore, it is very hard for her to find out the shared key.

3 Provably Authenticated Key Exchange Protocol

As the public keys of A and B described in Subsection 2.1 are not authenticated, the key exchange scheme is vulnerable to man-in-the-middle attack. If E(ve) substitutes A (or B)'s public key with her public key, then the protocol fails.

This section describes an authenticated key exchange protocol in which each party's public key is authenticated and sent to the other party. By using the authenticated public key, a secure scheme for key exchange can be obtained. This proposed protocol is based on the scheme described in paper [13]. A formal proof of the correctness of the proposed scheme based on BAN authentication logic [4] is also given here. The notations used in this protocol are listed in Table 1.

Table 1: Notations

Notations	Description
$x \in B_{l+r}$	A sufficiently complicated $(l+r)$-braid;
$\alpha_A \in LB_l$	A long-term private key of A, where LB_l is a sub group of B_{l+r} and generated by $\sigma_1, \cdots, \sigma_{l-1}$;
$\alpha_B \in LB_r$	A long-term private key of B, where LB_r is a sub group of B_{l+r} and generated by $\sigma_{l+1}, \cdots, \sigma_{l+r-1}$;
$P_A = \alpha_A x \alpha_A^{-1}$	A long-term public key of A;
$P_B = \alpha_B x \alpha_B^{-1}$	A long-term public key of B;
$\beta_A \in LB_l$	A session private key of A;
$\beta_B \in LB_r$	A session private key of B;
$p_A = \beta_A x \beta_A^{-1}$	A session public key of A;
$p_B = \beta_B x \beta_B^{-1}$	A session public key of B;
K_{AB}	A long-term common secret shared by A and B;
k_{AB}	A shared session key of A and B.

The steps of the protocol are as follows.

(a) A uses α_A and P_B to compute
$$\begin{aligned}K_{AB} &= \alpha_A\ P_B\ \alpha_A^{-1}\\ &= \alpha_A\ \alpha_B\ x\ \alpha_B^{-1}\ \alpha_A^{-1}.\end{aligned}$$

(b) B uses α_B and P_A to compute
$$\begin{aligned}K_{AB} &= \alpha_B\ P_A\ \alpha_B^{-1}\\ &= \alpha_B\ \alpha_A\ x\ \alpha_A^{-1}\ \alpha_B^{-1}.\\ &= \alpha_A\ \alpha_B\ x\ \alpha_B^{-1}\ \alpha_A^{-1}.\end{aligned}$$

(c) A computes authenticated public key $Ap_A = K_{AB}(\beta_A x \beta_A^{-1})K_{AB}^{-1}$ and sends it to B.

(d) Upon receiving Ap_A, B uses K_{AB} which he has computed in step (b) to get $p_A = \beta_A x \beta_A^{-1}$. B can then compute the shared session key $k_{AB} = \beta_B \beta_A x \beta_A^{-1} \beta_B^{-1} = \beta_A \beta_B x \beta_B^{-1} \beta_A^{-1}$.

(e) In a similar way to Steps (c) and (d), when A receives $Ap_B = K_{AB}(\beta_B x \beta_B^{-1})K_{AB}^{-1}$ from B, she can also compute the shared session key, $k_{AB} = \beta_A \beta_B x \beta_B^{-1} \beta_A^{-1}$.

According to the described scheme, E(ve) may intercept both Ap_A and Ap_B but she can substitute neither p_A nor p_B with her public key because she does not know K_{AB}. Therefore, the proposed scheme is considered to be immune to man-in-the-middle attack.

3.1 Proof of Authenticated Public Key

In the authenticated two-party key exchange scheme described above, each user has to believe the session public key received from the other party. Therefore, a proof of authenticated public key, which B has received from A, is done using BAN logic [4]. The notations used in this paper follow those of BAN logic.

In the above protocol, A sends an authenticated public key to B, i.e. $A \rightarrow B : K_{AB}(p_A)K_{AB}^{-1}$. The message can be transformed into the idealized form as $A \rightarrow B : \{p_A\}_{K_{AB}}$.

The goal is to prove that B believes p_A. To analyze the protocol, the following assumptions are made. B believes $A \overset{K_{AB}}{\rightleftarrows} B$, B believes fresh (p_A), B believes A controls (p_A). The steps of the proof are as follows:

1) B believes $A \overset{K_{AB}}{\rightleftarrows} B$ and B sees $\{p_A\}_{K_{AB}}$,then B believes A said p_A.

2) B believes fresh (p_A) and B believes A said p_A, then B believes A believes p_A.

3) B believes A controls p_A and B believes A believes p_A,then B believes p_A.

In the similar way, we can obtain the proof that A also believes p_B.

4 Security Analysis of Some Group Key Agreement Protocols

In the paper [1], an authenticated group key agreement protocol based on braid groups has been proposed. However, the number of rounds needed in the paper is linear to the number of group members. In each round some calculated values have to be sent from user i to user $i+1$ for further calculation. Therefore, the scheme can suffer from some long delays at some participants in the group.

In the paper [8], Hwang et al. has proposed a group key exchange scheme which needs only two rounds. The protocol is supposed to be scalable. However, in paper [14], Lee et al. has shown that Hwang et al.'s scheme does not provide forward and backward secrecy in dynamic environment. Lee et al. also gives an improvement of the scheme to remedy the problems.

In the following subsections, a brief description of Lee et al.'s scheme and a demonstration that the scheme does not preserve backward secrecy is given. Here, backward secrecy means that a new member should not be able to decrypt the multicast data sent before his joining, and forward secrecy means that a former member should not be able to decrypt the multicast data sent after his leaving [5]. In addition, security analysis of other two works [7, 12] using ring structure of participants is also discussed.

4.1 Lee et al.'s Scheme

In Lee et al.'s scheme [14], when users $U_1, \cdots, U_i, \cdots, U_n$ want to establish a secure communication, each member U_i performs the secure Diffie-Hellman two-party key exchange with his/her neighbors U_{i-1} and U_{i+1} and then negotiates the shared keys $k_{U_{i-1}U_i}$ and $k_{U_iU_{i+1}}$ as shown in Figure 1. It should be noted that, $k_{U_nU_1}$ is negotiated by U_n and U_1. Each user U_i then computes a value $Z_i = k_{U_{i-1}U_i} \oplus k_{U_iU_{i+1}}$ and broadcasts this value to other members. Note that, Z_n is computed as $k_{U_{n-1}U_n} \oplus k_{U_nU_1}$. Upon receiving all Z_j, where $j \neq i$ from other members, each user U_i can compute the other members' shared keys inductively as follows:

$$\begin{aligned} k_{U_{i+1}U_{i+2}} &= Z_{i+1} \oplus k_{U_iU_{i+1},} \\ k_{U_{i+2}U_{i+3}} &= Z_{i+2} \oplus k_{U_{i+1}U_{i+2},} \\ \vdots \quad & \quad \vdots \\ k_{U_nU_1} &= Z_n \oplus k_{U_{n-1}U_n,} \\ k_{U_1U_2} &= Z_1 \oplus k_{U_nU_1,} \\ \vdots \quad & \quad \vdots \\ k_{U_{i-2}U_{i-1}} &= Z_{i-2} \oplus k_{U_{i-3}U_{i-2},}. \end{aligned}$$

Each user U_i can then compute group shared key sk as $sk = H_0$ $(k_{U_1U_2} \shortparallel k_{U_2U_3} \cdots \shortparallel k_{U_{n-1}U_n} \shortparallel k_{U_nU_1})$, where $H_0()$ is a public one-way hash function from $\{1,0\}^*$ to $\{1,0\}^q$, where q is a security parameter.

The above scheme needs only two rounds to achieve the group shared key. However, when a new member joins the group, the scheme experiences a number of issues. In the protocol, when a new member U_{n+1} joins the group, he/she has to perform a two-party key exchange with his neighbors U_n and U_1 to obtain $k_{U_nU_{n+1}}$ and $k_{U_{n+1}U_1}$. Then new Z_n, Z_{n+1} and Z_1 of users U_n, U_{n+1} and U_1 are computed as $Z_n = k_{U_{n-1}U_n} \oplus k_{U_nU_{n+1}}$, $Z_{n+1} = k_{U_nU_{n+1}} \oplus k_{U_{n+1}U_1}$ and $Z_1 = k_{U_{n+1}U_1} \oplus k_{U_1U_2}$, respectively. All users then broadcast their computed Z values. Upon receiving all Z values, each user computes $k_{U_1U_2}$, $k_{U_2U_3}$, $\cdots$, $k_{U_{n-1}U_n}$, $k_{U_nU_{n+1}}$, $k_{U_{n+1}U_1}$. Finally each user can compute the new session key as $sk = H_0$ $(k_{U_1U_2} \shortparallel k_{U_2U_3} \cdots \shortparallel k_{U_{n-1}U_n} \shortparallel k_{U_nU_{n+1}} \shortparallel k_{U_{n+1}U_1})$.

The flaw in this scheme is that a new member can use the calculated $k_{U_1U_2}$, $k_{U_2U_3}$, $\cdots$, $k_{U_{n-1}U_n}$ which are the same as in the previous session and the value of the previous Z_n denoted by $(Z_n)_p$ which he/she saw in the previous session to compute $k_{U_nU_1} = (Z_n)_p \oplus k_{U_{n-1}U_n}$ of the previous session. Therefore, the new member can compute the shared key in the previous session as well.

A simple way to remedy the flaw is to make each member of the new group start the scheme from the beginning step of two-party key exchange with his/her neighbors when a new member joins the group. This should be done in the same way as the leave protocol described in Lee et al.'s work [14]. However, this can generate a lot of exchanged and broadcast messages in the network. This problem also arises when several groups want to merge together. To prevent such problems, a scheme which generates fewer messages and lessen users' computation is given in Section 5.

4.2 Dutta and Barua' Scheme

In the join algorithm of this scheme [7], a seed $x = H(sk)$, where sk is the group key of n users in the previous session and $H()$ is a hash function, is used for generating the new group key in the join session. A ring of U_1, U_2, U_n and a new user U_{n+1} is formed. In this ring, U_1 uses x_1, U_2 uses x, U_n uses x_n and U_{n+1} uses x_{n+1} as their private keys to perform key exchange and compute new group key. This scheme is vulnerable to known session key attack because an adversary who knows the group key in the previous session (in the case that he/she calls Function *reveal*() as described in paper [3]) can compute x and then shared keys between U_2 and his/her neighbors. The adversary can eventually compute the new group key.

In the leave algorithm, the remaining members in the group form a new ring. Only left-right neighbors of the leaving users choose new session private keys, and perform key exchange in order to establish new shared keys. Other members in the ring use their precomputed shared keys obtained in the previous session. These members of the new ring then use their shared keys to compute the new session group key. Since the leaving users also know these precomputed shared keys in the previous session, they can compute the new group key. Thus this scheme cannot preserve forward secrecy.

4.3 Kumar and Tripathi' Scheme

In the join algorithm of this scheme [12], a ring of U_1, U_n and U_{n+1} is formed to compute a shared key K. U_1 then encrypts K by using the previous session group key SK and broadcasts the encrypted message to all other members of the old group. Upon receiving this K valve, all members compute the new group key as $SK_{new} = H(SK \parallel K)$. Therefore, this scheme is vulnerable to

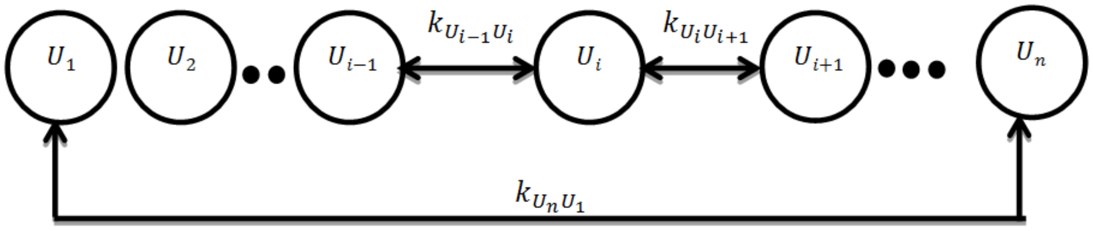

Figure 1: The structure of Lee et al.'s group key agreement

known session key attack because an adversary who knows the group key in the previous session can recover K and then compute the new group key.

Similarly to Dutta and Barua' leave algorithm, Kumar and Tripathi' one also takes advantage of some users' precomputed shared keys obtained in the previous session to compute the new session group key. Since the leaving users also know these precomputed shared keys in the previous session, they can compute the new group key. Thus this scheme does not preserve forward secrecy.

5 Authenticated Group Key Agreement Based on Braid Groups

In this section, the authenticated key exchange scheme described in Section 3 is extended so that it can be used in an authenticated group key agreement.

Suppose there are n subgroups $B_{l_1}, B_{l_2}, \cdots, B_{l_n}$ of l-braid group B_l, where $l = l_1 + l_2 + \cdots + l_n$. Let $U_1 \cdots U_i \cdots U_n$ be n users participating in the group communication protocol. Two complicated braids x and $x_1 \in B_l$ are published as system parameters. A long-term private key of each U_i is $\alpha_{U_i} \in B_{l_i}$ and the computed long-term public key is $P_{U_i} = \alpha_{U_i} x \alpha_{U_i}^{-1}$.

5.1 Group Initialization

Let $U_1 \cdots U_i \cdots U_m$, where $m \subset n$, be m users wishing to establish a secure group communication. These users are arranged in a predefined order of a ring. Each user U_i performs the secure authenticated two-party key exchange with his/her neighbors U_{i-1} and U_{i+1}. Note that U_m performs key exchange with U_{m-1} and U_1.

As described in Section 3, U_i and U_{i+1} can compute a shared secret $K_{U_iU_{i+1}} = \alpha_{U_i}\alpha_{U_{i+1}}\ x\ \alpha_{U_{i+1}}^{-1}\alpha_{U_i}^{-1}$ by using their long-term private and public keys. The steps of group initialization are as follows.

1) U_i chooses $x_{i,i+1} \in B_{l_i} \cup B_{l_{i+1}}$ and session private key $\beta_{U_i} \in B_{l_i}$, and computes a session public key $p_{U i} = \beta_{U_i}\ x_{i,i+1}\ \beta_{U_i}^{-1}$.

2) U_i sends $x_{i,i+1}$ and an authenticated public key $K_{U_iU_{i+1}}\ x_1\ (\beta_{U_i} x_{i,i+1}\ \beta_{U_i}^{-1})\ K_{U_iU_{i+1}}^{-1}$ to U_{i+1}. It is noted that x_1 is used for making $K_{U_iU_{i+1}}\ x_1\ (\beta_{U_i} x_{i,i+1}\ \beta_{U_i}^{-1})\ K_{U_iU_{i+1}}^{-1}$ and $x_1\ (\beta_{U_i} x_{i,i+1}\ \beta_{U_i}^{-1})$ at the same length in order to be immune to Length−based attack [10].

3) U_{i+1} uses x_1 and the shared secret $K_{U_iU_{i+1}}$ to verify the message, and then obtains U_i's session public key $\beta_{U_i} x_{i,i+1}\ \beta_{U_i}^{-1}$.

4) U_{i+1} computes the shared key $k_{U_iU_{i+1}} = \beta_{U_{i+1}}\beta_{U_i} x_{i,i+1}\beta_{U_i}^{-1}\beta_{U_{i+1}}^{-1}$ by using his/her session private key $\beta_{U_{i+1}} \in B_{l_{i+1}}$.

5) In the same way as described above, U_{i+1} sends an authenticated public key $K_{U_iU_{i+1}}\ x_1\ (\beta_{U_{i+1}} x_{i,i+1}\ \beta_{U_{i+1}}^{-1})\ K_{U_iU_{i+1}}^{-1}$ to U_i. Upon receiving U_{i+1}'s authenticated public key, U_i can compute the shared key $k_{U_iU_{i+1}} = \beta_{U_i}\beta_{U_{i+1}} x_{i,i+1}\beta_{U_{i+1}}^{-1}\beta_{U_i}^{-1}$, since $\beta_{U_{i+1}}\beta_{U_i} = \beta_{U_i}\beta_{U_{i+1}}$.

6) In the same way, U_i and U_{i-1} can also compute the shared key $k_{U_{i-1}U_i} = \beta_{U_i}\beta_{U_{i-1}} x_{i-1,i}\beta_{U_{i-1}}^{-1}\beta_{U_i}^{-1}$, where $x_{i-1,i} \in B_{l_{i-1}} \cup B_{l_i}$, and $\beta_{U_{i-1}} \in B_{l_{i-1}}$ is U_{i-1}'s session private key.

7) Each user U_i computes a value $Z_{U_i} = (k_{U_{i-1}U_i})^{-1} k_{U_iU_{i+1}}$ and broadcasts the value to other members. Note that $Z_{U_m} = (k_{U_{m-1}U_m})^{-1} k_{U_mU_1}$.

8) When each user U_i obtains all Z_{U_j}, where $1 \leq j \leq m$ and $j \neq i$, from other members, he/she checks whether the received Z_{U_j} values come from the existing group members by computing

$$\begin{aligned} Z_0 &= Z_{U_1} Z_{U_2} \cdots Z_{U_m} \\ &= (k_{U_mU_1})^{-1} k_{U_1U_2} (k_{U_1U_2})^{-1} k_{U_2U_3} \cdots \\ &\quad (k_{U_{m-2}U_{m-1}})^{-1} k_{U_{m-1}U_m} (k_{U_{m-1}U_m})^{-1} k_{U_mU_1}. \end{aligned}$$

If all Z_{U_j} come from the existing group members, then the value Z_0 is equal to the identity braid. When a user finds out that Z_0 is not the identity braid, he/ she will broadcast an error message. Upon receiving the error message, each user halts the scheme.

9) If Z_0 is the identity braid, each user U_i which has $k_{U_{i-1}U_i}$ and $k_{U_iU_{i+1}}$, begins to compute the shared keys of other members as follows:

$$\begin{aligned} k_{U_{i+1}U_{i+2}} &= k_{U_iU_{i+1}} Z_{U_{i+1}} \\ &= k_{U_iU_{i+1}} (k_{U_iU_{i+1}})^{-1} k_{U_{i+1}U_{i+2}} \\ k_{U_{i+2}U_{i+3}} &= k_{U_{i+1}U_{i+2}} Z_{U_{i+2}} \\ &= k_{U_{i+1}U_{i+2}} (k_{U_{i+1}U_{i+2}})^{-1} k_{U_{i+2}U_{i+3}} \\ &. \\ &. \\ k_{U_{i-2}U_{i-1}} &= k_{U_{i-3}U_{i-2}} Z_{U_{i-2}} \\ &= k_{U_{i-3}U_{i-2}} (k_{U_{i-3}U_{i-2}})^{-1} k_{U_{i-2}U_{i-1}}. \end{aligned}$$

10) After collecting all shared keys, each user computes a seed $sd = k_{U_1U_2} k_{U_2U_3} \cdots k_{U_{m-1}U_m}$ and the group session key $sk_G = (sd)x(sd)^{-1}$. Note that $sd \in B_{l_1} \cup B_{l_2} \cup \cdots \cup B_{l_m}$ and $sk_G \in B_l$.

Complexity: The group initialization needs two rounds. In the first round, $O(m)$ unicast messages are sent for key exchange. Each user computes $O(1)$ braid multiplication. In the second round, $O(m)$ broadcast messages are used for sending shared keys. Each user has to compute $O(m)$ braid multiplication in order to achieve the seed and the common group key.

5.2 Member Join

When a new user U_{m+1} wants to join the group, a user in the existing group can present himself/herself as the group representative U_r and performs key exchange with user U_{m+1} in order to construct a new group key.

Let $\alpha_{U_r} \in B_{l_r}$ and $P_{U_r} = \alpha_{U_r} x \alpha_{U_r}^{-1} \in B_l$ be the long-term private key and public key of U_r. In addition $\alpha_{U_{m+1}} \in B_{l_{m+1}}$ and $P_{U_{m+1}} = \alpha_{U_{m+1}} x \alpha_{U_{m+1}}^{-1}$ be the keys of U_{m+1}. Note that Users U_r and U_{m+1} can establish a shared secret $K_{U_rU_{m+1}} = \alpha_{U_r} \alpha_{U_{m+1}} x \alpha_{U_{m+1}}^{-1} \alpha_{U_r}^{-1}$ with each other. The steps needed to establish a new group key are as follows.

(a) User U_r chooses a braid $x_{G,m+1} \in B_{l_1} \cup B_{l_2} \cup \cdots B_{l_m} \cup B_{l_{m+1}}$ and sends it together with the authenticated blind seed $K_{U_rU_{m+1}} x_1 (sd) x_{G,m+1} (sd)^{-1} K_{U_rU_{m+1}}^{-1}$ to U_{m+1}. It is noted that sd is the seed obtained in the previous session.

(b) Upon receiving the authenticated blind seed, U_{m+1} uses $K_{U_rU_{m+1}}$ and x_1 to recover $(sd)x_{G,m+1}(sd)^{-1}$. and computes the seed of the new group as

$$sd_{nG} = \beta_{U_{m+1}} (sd) x_{G,m+1} (sd)^{-1} \beta_{U_{m+1}}^{-1},$$

where $\beta_{U_{m+1}} \in B_{l_{m+1}}$ is user U_{m+1}'s session private key. U_{m+1} then computes the new group key sk_{nG} $=(sd_{nG})x(sd_{nG})^{-1}$.

(c) In a similar way, U_r can compute the seed of the new group $sd_{nG} = (sd)\beta_{U_{m+1}} x_{G,m+1} \beta_{U_{m+1}}^{-1} (sd)^{-1}$ by using the value $K_{U_rU_{m+1}} x_1 (\beta_{U_{m+1}} x_{G,m+1} \beta_{U_{m+1}}^{-1})\ K_{U_rU_{m+1}}^{-1}$ received from U_{m+1}. Then U_r uses the seed to compute the new group key sk_{nG}.

(d) U_r broadcasts $(sd)x_1 sd_{nG}\ x_1^{-1}(sd)^{-1}$ to all members of the previous group.

(e) Upon receiving the broadcast value, each user of the previous group uses the previous seed sd and x_1 to recover the new seed sd_{nG} and then computes the new group key sk_{nG}.

In the case that many users, e.g. U_{m+1}, U_{m+2} and U_{m+3} want to join the group simultaneously, U_r can use the seed sd of the old group to establish an intermediate group with U_{m+1}, U_{m+2} and U_{m+3} by adopting the group initialization described in Subsection 5.1. After computing the new seed sd_{nG}, U_r can use the scheme described above to send the new seed to other members of the old group.

Complexity: When there are m' users want to join the group, this algorithm needs three rounds. In the first round, $O(m')$ unicast messages are sent for key exchange. Each user computes $O(1)$ braid multiplication. In the second round, $O(m')$ broadcast messages are used for sending users' shared keys. Each of $m'+1$ users has to compute $O(m')$ braid multiplication in order to achieve the new seed and group key. In the third round, one broadcast message is sent from the representative to other members in the old group. Each member computes $O(1)$ braid multiplication in order to achieve the new seed.

5.3 Member Leave

When a member leaves a group of m members, each remaining member can choose a new session private key and starts the protocol with his/her neighbors from the step containing the two-party key exchange, to construct a new session key for $m-1$ members in the way described in Subsection 5.1. By performing such scheme, forward secrecy is preserved. However, this scheme generates many messages in the network. A scheme which can reduce communicating messages is described below.

Let us consider a scenario where there are 6 members in the existing group, and user U_4 wants to leave the group. The steps of the leave algorithm are as follows.

(a) U_3 performs an authenticated two-party key exchange with user U_5 to obtain a shared key $k_{U_3U_5} = \beta_{U_3}\beta_{U_5}$ $x_{3,5}\ \beta_{U_5}^{-1}\beta_{U_3}^{-1}$, where $x_{3,5} \in B_{l_3} \cup B_{l_5}$, and β_{U_3}, β_{U_5} are U_3 and U_5 session private keys, respectively.

(b) U_3 and U_5 computes a new seed $sd_{nG} = k_{U_3U_5}$ $sd(k_{U_3U_5})^{-1}$ where sd is the seed of the old group. They then compute the new group key sk_{nG} $=(sd_{nG})x(sd_{nG})^{-1}$.

(c) U_3 then uses authenticated messages $(K_{U_3U_i})$ (sd) $x_1 sd_{nG} x_1^{-1}$ $(sd)^{-1}$ $(K_{U_3U_i})^{-1}$ to send the seed to the remaining members U_i in the group, where $K_{U_3U_i} = \alpha_{U_3}\alpha_{U_i}x\alpha_{U_i}^{-1}\alpha_{U_3}^{-1}$, i= 1,2,6.

(d) Upon receiving the authenticated message, U_i uses $K_{U_3U_i}$, sd and x_1 to recover sd_{nG}.

In the case that $m^{'}$ users leave the group simultaneously, left-right neighbors of the leaving users form an intermediate group in order to compute a shared secret among them. They then use this secret and the seed obtained in the previous session to compute the new seed, and send the authenticated seed to the remaining members in the group.

Complexity: The leave algorithm needs three rounds in case that there are $m^{'}$ users leave a group of m users. In the first round, $O(m^*)$ unicast messages, where m^* is $\min(m - m^{'}, m^{'})$, are sent for establishing key exchange between the left-right neighbors of the leaving users. In the second round, $O(m^*)$ broadcast messages are sent and each participating user computes $O(m^*)$ braid multiplication in order to achieve the new seed and group key. In the third round, $O(m - m^{'})$ unicast messages are used for sending the new seed to the remaining members in the group. Upon receiving the message, each user has to compute $O(1)$ braid multiplication in order to recover the seed and compute the new group key.

5.4 Group Merge

The join protocol described in Subsection 5.2 can be extended for merging two or more groups together. The idea is that the representative of each group uses the existing seed to perform two-party key exchange with each other to establish a new seed and group key. Then the representative of each group sends the new seed to other members of the group. In this subsection, an example of the merging scheme for two groups $G1$ and $G2$ is described.

For simplicity, let $U_1 \cdots U_{r_{G1}} \cdots U_m$ be members of $G1$ and $U_{m+1} \cdots U_{r_{G2}} \cdots U_n$ be members of $G2$. $U_{r_{G1}}$ and $U_{r_{G2}}$ are users who claim to be representatives of group $G1$ and $G2$, respectively. Let $sd_{G1} \in B_{l_1} \cup B_{l_2} \cup \cdots \cup B_{l_m}$ and $sd_{G2} \in B_{l_{m+1}} \cup B_{l_{m+2}} \cup \cdots \cup B_{l_n}$ be the seeds of $G1$ and $G2$, respectively. The steps of the protocol are as follows.

(a) $U_{r_{G1}}$ and $U_{r_{G2}}$ compute a shared secret $K_{U_{rG1}U_{rG2}} \in B_l$ by using their long-term private and public keys as described in Section 3.

(b) $U_{r_{G1}}$ chooses a braid $x_{G1,G2} \in B_{l_1} \cup B_{l_2} \cup \cdots \cup B_{l_m} \cup B_{l_{m+1}} \cup \cdots \cup B_{l_{n-1}} \cup B_{l_n}$ and sends it together with the authenticated blind seed $K_{U_{rG1}U_{rG2}}$ x_1 (sd_{G1}) $x_{G1,G2}$ $(sd_{G1})^{-1}$ $(K^{-1}_{U_{rG1}U_{rG2}})$ to U_{rG2}.

(c) Upon receiving the authenticated blind seed, U_{rG2} computes the new seed $sd_{nG} = (sd_{G2})$ (sd_{G1}) $x_{G1,G2}$ $(sd_{G1})^{-1}$ $(sd_{G2})^{-1}$ and the new group key $sk_{nG} = (sd_{nG})$ x $(sd_{nG})^{-1}$.

(d) U_{rG2} sends the new seed to the remaining members of $G2$ by using the method described in Subsection 5.2.

(e) In a similar way, $U_{r_{G1}}$ computes the new seed and common group key by using the authenticated blind seed received from $G2$ and sends the new seed to the remaining members of $G1$.

Everyone in the merging group can now securely communicate by using the new session group key with backward secrecy.

Complexity: When there are J groups which want to merge together, this algorithm needs three rounds. In the first round, $O(J)$ unicast messages are sent by group representatives for key exchange. Each representative computes $O(1)$ braid multiplication in order to obtain shared keys. In the second round, $O(J)$ broadcast messages are used for sending shared keys. Each group representative has to compute $O(J)$ braid multiplication in order to achieve the new seed and group key. In the third round, one broadcast message is sent from each group representative to the remaining members in the group. The remaining members of each group computes $O(1)$ braid multiplication in order to recover the new seed.

5.5 Group Partition

A group can be partitioned into two or more groups. Group partition scheme can be done by letting all members of each new group work together to establish a new group key, by adopting the group initialization scheme described in Subsection 5.1 and using a new session private key for each user.

In this subsection, the extension of the leave protocol for group partition is described. For simplicity, we consider a scenario in which there are 10 members in the existing group, The group are then partitioned into two groups. The first one has seven users, i.e. U_1, U_2, U_5, U_6, U_7, U_9 and U_{10}. Users U_3, U_4 and U_8 are members of the second group. Let us consider the scenario of the first group which U_3, U_4 and U_8 leave the group. The left-right neighbors of the leaving users, i.e. U_2, U_5, U_7 and U_9 then choose new session private keys and work together to establish an intermediate seed sd_{iG} by using the group initialization scheme. Each user of the intermediate group then computes a new seed $sd_{nG} = sd_{iG}\ sd(sd_{iG})^{-1}$, where sd is the seed obtained in the previous session. U_5 which is the right-handed neighbor of the leaving user U_4 then sends the authenticated seed, as described in 5.3, to U_6 which is the remaining group member on the right-handed side of U_5. U_9 also sends the authenticated seed to U_{10} and U_1 which are the remaining group members on the right-handed side of U_9. As the leaving users do not know sd_{iG}, they cannot compute sd_{nG}. Therefore this scheme can preserve forward secrecy. U_3, U_4 and U_8 can also compute the new seed and shared key of their group by using the same approach.

Complexity: Suppose a group of m users is partitioned into J groups, each new group has m_j members, where

j =1 to J. The partition algorithm needs three rounds for each group. In the first round, $O(m_j^*)$ unicast messages, where m_j^* is $\min(m - m_j, m_j)$, are sent for two-party key exchange between participating users in each group. In the second round, $O(m_j^*)$ broadcast messages are sent, and each participating user in each group computes $O(m_j^*)$ braid multiplication in order to achieve the new seed and group key. In the third round, $O(m_j)$ unicast messages are used for sending the new seed to the remaining members in each group. Upon receiving the message, each user has to compute $O(1)$ braid multiplication in order to recover the seed and compute the group key.

6 Authentication Proof of the Group Key Agreement

In this section, an authentication proof of the proposed group-key-agreement protocol is shown using BAN Logic. This proof is different from the authentication proof given in Lee et al′s work [14]. The notations used in this work follow those in paper [4]. As described in Subsection 5.1, each user U_i can compute a shared secret (key) with his/her neighbors, i.e. U_i believes $U_i \overset{k_{U_iU_{i+1}}}{\rightleftarrows} U_{i+1}$ and U_i believes $U_i \overset{k_{U_{i-1}U_i}}{\rightleftarrows} U_{i-1}$.

Further, when each user U_i receives all Z values from other members, he/she uses these values to compute $k_{U_1U_2}$, $k_{U_2U_3}$, $\cdots$, $k_{U_{m-1}U_m}$ and uses them to compute the session seed and group key.

The aim of this proof is to thus show that U_i can believe $k_{U_1U_2}$ because it is the shared secret (key) between U_1 and U_2 in the existing group, and U_i can believe $k_{U_2U_3}$ because it is the shared secret (key) between U_2 and U_3, and so on. U_i can then use these values to compute the seed and group key. This means that the aim of the proof is to show that the following statement is satisfied under the proposed group-key-agreement protocol:

U_i believes ($U_1 \overset{k_{U_1U_2}}{\rightleftarrows} U_2$, $U_2 \overset{k_{U_2U_3}}{\rightleftarrows} U_3$, $\cdots$, $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$, $\cdots$, $U_{m-1} \overset{k_{U_{m-1}U_m}}{\rightleftarrows} U_m$).

At the group initialization stage, each user U_i receives the broadcast messages Z from other members of the group. These messages can be transformed into the idealized forms as follows:

$$U_{i+1} \to U_i : \{N_{i+1},\, U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}\}_{k_{U_iU_{i+1}}}.$$

$$U_{i+2} \to U_i : \{N_{i+2},\, U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}\}_{k_{U_{i+1}U_{i+2}}}.$$

$$U_{i+3} \to U_i : \{N_{i+3},\, U_{i+3} \overset{k_{U_{i+3}U_{i+4}}}{\rightleftarrows} U_{i+4}\}_{k_{U_{i+2}U_{i+3}}}.$$

.

.

$$U_{i-1} \to U_i : \{N_{i-1},\, U_{i-1} \overset{k_{U_{i-1}U_i}}{\rightleftarrows} U_i\}_{k_{U_{i-2}U_{i-1}}}.$$

In this form $N_1, N_2, \cdots, N_{i+1}, N_{i+2}, \cdots, N_m$ are nonces.

To analyze the protocol, the following assumptions are made.

U_i believes fresh $(N_1, N_2, \cdots, N_{i+1}, N_{i+2}, \cdots, N_m)$

U_i believes U_{i+1} controls $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$,

U_i believes U_{i+2} controls $U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$,

.

.

U_i believes U_{i-1} controls $U_{i-1} \overset{k_{U_{i-1}U_i}}{\rightleftarrows} U_i$.

The main steps for the proof are as follows:

1) U_i believes $U_i \overset{k_{U_iU_{i+1}}}{\rightleftarrows} U_{i+1}$ and U_i sees $\{N_{i+1}, U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}\}_{k_{U_iU_{i+1}}}$, then U_i believes U_{i+1} said (N_{i+1}, $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$).

2) U_i believes fresh (N_{i+1}) and U_i believes U_{i+1} said (N_{i+1}, $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$), then U_i believes U_{i+1} believes (N_{i+1}, $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$).

   The conjunction can be broken and this yields the result that

   U_i believes U_{i+1} believes $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$.

3) U_i believes U_{i+1} controls $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$ and U_i believes U_{i+1} believes $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$, then U_i believes $U_{i+1} \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$.

   The above conclusion means that U_i can believe $k_{U_{i+1}U_{i+2}}$ which may be used by U_{i+2} to send a message. This can be written in the form

   U_i believes $U_i \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$.

4) U_i believes $U_i \overset{k_{U_{i+1}U_{i+2}}}{\rightleftarrows} U_{i+2}$ and U_i sees $\{N_{i+2}, U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}\}_{k_{U_{i+1}U_{i+2}}}$, then U_i believes U_{i+2} said (N_{i+2}, $U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$).

5) U_i believes fresh (N_{i+2}) and U_i believes U_{i+2} said (N_{i+2}, $U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$,then U_i believes U_{i+2} believes (N_{i+2}, $U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$).
   The conjunction can be broken and this yields the result that U_i believes U_{i+2} believes $U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$.

6) U_i believes U_{i+2} controls $U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$ and U_i believes U_{i+2} believes $U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$, then U_i believes $U_{i+2} \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$. The above conclusion means that U_i can believe $k_{U_{i+2}U_{i+3}}$ which may be used by U_{i+3} to send a message. This can be written in the form U_i believes $U_i \overset{k_{U_{i+2}U_{i+3}}}{\rightleftarrows} U_{i+3}$. By repeating the steps above, we obtain the following result. U_i believes ($U_1 \overset{k_{U_1U_2}}{\rightleftarrows} U_2$, $U_2 \overset{k_{U_2U_3}}{\rightleftarrows} U_3, \cdots, U_{i+3} \overset{k_{U_{i+3}U_{i+4}}}{\rightleftarrows} U_{i+4}, \cdots, U_{m-1} \overset{k_{U_{m-1}U_m}}{\rightleftarrows} U_m$).

Therefore we have shown that the statement is satisfied under the proposed protocol, and U_i can use these shared keys to compute the session seed and group key.

In the member join protocol described in Subsection 5.2, the new group key is obtained by allowing U_r to perform an authenticated key exchange, using the seed obtained in the previous session, with the new member using his/her private key. U_r then broadcasts the new seed to other members of the former group using the old seed which is known only to members of the group. Therefore, all members of the former group believe received seed and use the seed to compute the new group key. This is similar to the merge protocol.

As for leave protocol, the left-handed neighbor of the leaving member sends the new seed to each member of the new group by authenticating and hiding the new seed in a secret known to only him/herself and each member. Therefore, each member of the new group believes the received seed and uses it to compute the new group key.

7 Security Analysis

In this section, security analysis of the proposed protocol is first discussed. Comparisons among ring-structure based protocols are then illustrated. In papers [7, 12], the proposed group- key-agreement protocols were analyzed by applying the toy game [3]. However, they still suffer some flaws because the communicating messages flowing in each session are not well analyzed. Here, we will emphasize on analyzing the messages exchanged in each session. As group initialization of the ring-structure protocols has been well discussed by Hwang et al. [8] as well as Dutta and Barua [7], security analysis of the algorithm is omitted. Here, we will discuss on join and leave algorithms. Merge and partition algorithms will not be discussed as they have similar features as join and leave algorithms respectively.

7.1 Analysis of Join Algorithm

As for single join, two types of massages are used. The first is exchanged messages between the representative of the existing group and the new user in order to compute the new seed. These are $K_{U_rU_{m+1}}x_1(sd)x_{G,m+1}(sd)^{-1}K^{-1}_{U_rU_{m+1}}$ and $K_{U_rU_{m+1}}x_1(\beta_{U_{m+1}}x_{G,m+1}\beta^{-1}_{U_{m+1}})\ K^{-1}_{U_rU_{m+1}}$. The second is the message $(sd)x_1sd_{nG}\ x_1^{-1}(sd)^{-1}$ sent by the representative to the remaining members of the group. The new user can analyze $(sd)x_{G,m+1}(sd)^{-1}$ and $(sd)x_1sd_{nG}\ x_1^{-1}(sd)^{-1}$. However, he/she cannot compute sd though he/she knows x_1, $x_{G,m+1}$ and sd_{nG}, because of the hardness of GCSP. Therefore the algorithm does preserve backward secrecy.

If an adversary can know the group key of the previous session, i.e.$(sd)x(sd)^{-1}$, he/she still cannot compute sd because of the hardness of GCSP. Therefore he/she cannot compute the new seed. The algorithm is thus immune to known session key attack.

If an adversary can know the long-term private keys of the representative and/ or of the new user, he/she can compute $K_{U_rU_{m+1}}$ and can see $x_1(sd)x_{G,m+1}(sd)^{-1}$ However, he/she cannot compute for sd. Therefore the algorithm preserves perfect forward secrecy.

7.2 Analysis of Leave Algorithm

In the single−leave algorithm, authenticated public keys are exchanged between the left neighbor U_{lt}, and the right one U_{rt} of the leaving user in order to compute new shared key $k_{U_{lt}U_{rt}}$ between them. The new seed of the group is computed as $sd_{nG} = k_{U_{lt}U_{rt}}\ sd(k_{U_{lt}U_{rt}})^{-1}$, where sd is the seed of the old group. U_{lt} then sends the new seed to each remaining member of the group by using the message $(K_{U_{lt}U_i})\ (sd)\ x_1sd_{nG}\ x_1^{-1}(sd)^{-1}\ (K_{U_{lt}U_i})^{-1}$, where $K_{U_{lt}U_i} = \alpha_{U_{lt}}\alpha_{U_i}\ x\alpha^{-1}_{U_i}\ \alpha^{-1}_{U_{lt}}$.

Each remaining member of the group uses $K_{U_{lt}U_i}$, sd and x_1 to extract the new seed. Although the leaving user knows sd and x_1, he /she cannot compute sd_{nG} because he/she knows neither $k_{U_{lt}U_{rt}}$ nor $K_{U_{lt}U_i}$. Therefore the algorithm does preserve forward secrecy.

If an adversary can know the group key $(sd)x(sd)^{-1}$ in the previous session, he/she still cannot compute sd because of the hardness of GCSP. Therefore the algorithm is immune to known session key attack.

If an adversary can know the long-term private keys of U_{lt} and/or U_i, he/she can compute $K_{U_{lt}U_i}$, and can see $(sd)\ x_1\ sd_{nG}\ x_1^{-1}\ (sd)^{-1}$. However, he/she can compute neither sd nor sd_{nG}. Therefore the algorithm preserves perfect forward secrecy.

7.3 Comparisons among Ring-Structure based Protocols

Table 2 illustrates comparisons of security and complexity among several ring-structure based protocols. According to the comparisons, we can see that the proposed protocol outperforms other protocols.

Table 2: Comparison table

Protocol	Authentication Techniques	NoR	**Group Operation**	Comment	**NoU**	**NoB**	**NoO**
Hwang et al. [8]	Not mentioned	*	Initialization	Yes	$O(m)$	$O(m)$	$O(m)$
Dutta&Barua [7]	Signatures	1	Initialization	Yes	$O(m)$	$O(m)$	$O(m)$
			Join	C2	*	*	*
			Leave	C4	*	*	*
Lee et al. [14]	Not mentioned	*	Initialization	Yes	$O(m)$	$O(m)$	$O(m)$
			Join	C3	*	*	*
			Leave	Yes	$O(m)$	$O(m)$	$O(m)$
Kumar&Tripathi [12]	Signatures	1	Initialization	Yes	$O(m)$	$O(m)$	$O(m)$
			Join	C2	*	*	*
			Leave	C4	*	*	*
Zhu [21]	Long-term keys &Session keys	4	Initialization	Yes	$O(m)$	$O(m)$	$O(m)$
			Join	Yes	O$(m^{'})$ $m^{'}=1$	$O(m)$	$O(m)$
			Leave	Yes	O$(m^{'})$ $m^{'}=1$	$O(m)$	$O(m)$
Proposed Protocol	Long-term keys	1	Initialization	Yes	$O(m)$	$O(m)$	$O(m)$
			Join	Yes	$O(m^{'})$	$O(m^{'})$	$O(m^{'})$
			Leave	Yes	$O(m-m^{'})$	$O(m^*)$	$O(m^*)$
			Merge	Yes	$O(J)$	$O(J)$	$O(J)$
			Partition	Yes	$O(m_j)$	$O(m_j^*)$	$O(m_j^*)$

NoR: number of rounds.
NoU: number of unicast messages.
NoB: number of broadcast messages.
NoO: number of operations.
C2: the protocol is vulnerable to known session key attack.
C3: the protocol does not preserve backward secrecy.
C4: the protocol does not preserve forward secrecy.
*** :** it is not discussed because this algorithm has some flaws or it is not mentioned.
Join: a group of m users becomes a group of $m+m^{'}$ users
Leave: $m^{'}$ users leave from a group of m users. $m^* = min(m^{'}, m-m^{'})$
Merge: J groups merge into one group
Partition: one group of m users becomes J groups, each group has m_j users, $m_j^* = min(m_j, m-m_j)$.
In the protocol, NoU, NoB and NoO are considered for each group

8 Conclusion

In this paper, braid group cryptography which requires only multiplication operations is adopted to establish a session group key. In order to prevent a man-in-the-middle attack, exchanged messages are authenticated using long-term private and public keys of group members. The proposed scheme is a scalable one. It needs only two rounds for initializing a group. In dynamic case, the scheme needs three rounds but with only few users involved. According to the comparisons among ring-structure based protocols, our protocol outperforms several protocols. An authentication proof is also shown in this paper using the well-known BAN logic. Although the proposed protocol is based on braid group cryptography, the framework can be applied to several cryptosystems including Diffie-Hellman, Elliptic Curve and chaotic maps.

References

[1] T. Aneksrup and P. Hiranvanichakorn, "Efficient group key agreement on tree-based braid groups," *Computer and Information Science*, vol. 4, no. 1, pp. 14–27, 2011.

[2] E. Artin, "Theory of braids," *Annals of Mathematics*, vol. 48, no. 1, pp. 101–126, 1947.

[3] E. Bresson, O. Chevassut, and D. Pointcheval, "Provably authenticated group diffie-hellman key exchange - The dynamic case," *Proceedings of Advances on Cryptology (Asiacrypt'01)*, pp. 290–309, 2001.

[4] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Transactions on Computer Systems*, vol. 8, no. 1, pp. 18–36, 1990.

[5] K. C. Chan and S. H. Chan, "Key management approaches to offer data confidentiality for secure multicasts," *IEEE Network*, vol. 17, no. 5, pp. 30–39, 2003.

[6] A. Chaturvedi and S. Lal, "An authenticated key agreement protocol using conjugacy problem in braid groups," *International Journal of Network Security*, vol. 6, no. 2, pp. 181–184, 2008.

[7] R. Dutta and R. Barua, "Provably secure constant round contributory group key agreement in dynamic setting," *IEEE Transactions on Information Theory*, vol. 54, no. 5, pp. 2007–2025, 2008.

[8] J. Y. Hwang, S. M. Lee, and D. H. Lee, "Scalable key exchange transformation: from two party to group," *Electronic Letters*, vol. 40, no. 12, pp. 728–729, 2004.

[9] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security*, vol. 7, no. 1, pp. 60–96, 2004.

[10] K. H. Ko, J. W. Lee, and T. Thomas, "Towards generating secure keys for braid cryptography," *Designs, Codes and Cryptography*, vol. 45, no. 3, pp. 317–333, 2007.

[11] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J. S. Kang, and C. Park, "New public key cryptosystem using braid groups," *Proceedings of Advances on Cryptography (Crypto'00)*, pp. 166–183, 2000.

[12] A. Kumar and S. Tripathi, "Anonymous ID-based group key agreement protocol without pairing," *International Journal of Network Security*, vol. 18, no. 2, pp. 263–273, 2016.

[13] H. K. Lee, H. S. Lee, and Y. R. Lee, "An authenticated group key agreement protocol on braid groups," *IACR Cryptology ePrint Archive*, 18, 2003. (`https://eprint.iacr.org/2003/018.pdf`)

[14] J. S. Lee, C. C. Chang, and K. J. Wei, "Provably secure conference key distribution mechanism preserving the forward and backward secrecy," *International Journal of Network Security*, vol. 16, no. 2, pp. 405–410, 2013.

[15] D. Li and S. Sampalli, "Group rekeying scheme for dynamic peer group security in collaborative networks," *International Journal of Network Security*, vol. 18, no. 5, pp. 946–959, 2016.

[16] V. S. Naresh and N. V. E. S. Murthy, "Elliptic curve based dynamic contributory group key agreement protocol for secure group communication over ad-hoc networks," *International Journal of Network Security*, vol. 17, no. 5, pp. 588–596, 2015.

[17] Q. Qian, Y. L. Jia, and R. Zhang, "A lightweight RFID security protocol based on elliptic curve cryptography," *International Journal of Network Security*, vol. 18, no. 2, pp. 354–361, 2016.

[18] R. S. Ranjani, D. L. Bhaskari, and P. S. Avadhani, "An extended identity based authenticated asymmetric group key agreement protocol," *International Journal of Network Security*, vol. 17, no. 5, pp. 510–516, 2015.

[19] G. K. Verma, "A proxy blind signature scheme over braid groups," *International Journal of Network Security*, vol. 9, no. 3, pp. 214–217, 2009.

[20] G. K. Verma, "Probable security proof of a blind signature scheme over braid groups," *International Journal of Network Security*, vol. 12, no. 2, pp. 118–120, 2011.

[21] H. F. Zhu, "Secure chaotic maps-based group key agreement scheme with privacy preserving," *International Journal of Network Security*, vol. 18, no. 6, pp. 1001–1009, 2016.

Biography

Pipat Hiranvanichakorn received the B.E. degree in Electrical Engineering from Chulalongkorn University, Thailand, in 1977 and the M.E. and D.E. degrees in Information Processing from Tokyo Institute of Technology, Japan, in 1982 and 1985. He is currently an associate professor of Computer Science at School of Applied Statistics, National Institute of Development Administration, Thailand. His current research interests include natural language processing, computer networks, cryptography and information security.