Provably-Secure and Communication-Efficient Scheme for Dynamic Group Key Exchange

Junghyun Nam Sungduk Kim Seungjoo Kim Dongho Won

May 17, 2004

School of Information and Communication Engineering,
Sungkyunkwan University, Korea
jhnam@dosan.skku.ac.kr, sdkim@koscom.co.kr, skim@ece.skku.ac.kr,
dhwon@simsan.skku.ac.kr

Abstract

Group key agreement protocols are designed to solve the fundamental problem of securely establishing a session key among a group of parties communicating over a public channel. Although a number of protocols have been proposed to solve this problem over the years, they are not well suited for a high-delay wide area network; their communication overhead is significant in terms of the number of communication rounds or the number of exchanged messages, both of which are recognized as the dominant factors that slow down group key agreement over a networking environment with high communication latency. In this paper we present a communication-efficient group key agreement protocol and prove its security in the random oracle model under the factoring assumption. The proposed protocol provides perfect forward secrecy and requires only a constant number of communication rounds for any of group rekeying operations, while achieving optimal message complexity.

Keywords: group key agreement, authenticated key agreement, provable security, factoring

1 Introduction

Group key agreement protocols enable a group of parties communicating over an open network to reach an agreement for a common secret key (called a *session key*). Typically, this session key is used to facilitate standard security services, such as confidentiality and data integrity, in numerous group-oriented applications including audio/video conferencing, distributed database, and various collaborative computing systems. In other words, the goal of group key agreement protocols is to efficiently implement secure group communication channels over untrusted, public networks. To this end, it is of prime importance for a group key agreement protocol to satisfy the property referred to as *implicit key authentication*, whereby each member is assured that no one other than the group members can obtain any information about the value of the session key. Therefore, as a result of the increased popularity of group-oriented applications, the design of an efficient authenticated group key agreement protocol has recently received much attention in the literature [4, 29, 15, 25, 10, 11].

Many problems related to group key agreement have been tackled and solved, especially over the last ten years, resulting in some constant-round protocols [25, 11] with provable security in concrete, realistic setting. However, all provably-secure protocols achieving forward secrecy so far are too expensive for dynamic groups, where current members may leave the group and new members may join the group at any time in an arbitrary manner. A group key agreement scheme for such a dynamic group must ensure that the session key is updated upon every membership change, so that subsequent communication sessions are protected from leaving members and previous communication sessions are protected from joining members. Although this can be achieved by running any authenticated group key agreement protocol from scratch whenever group membership changes, alternative approaches to handle this dynamic membership more efficiently would be clearly preferable. Indeed, several dynamic group key agreement schemes have been proposed to minimize the cost of the rekeying operations associated with group updates [1, 12, 13, 23, 24, 29].

1.1 Related Work

The original idea of extending the 2-party Diffie-Hellman scheme [17] to the multi-party setting dates back to the classical paper of Ingemarsson et al. [21], and is followed by many works [27, 16, 22, 4, 23, 3, 29, 30, 24] offering various levels of complexity. However, regardless of whether they explicitly deal with the case where group membership is dynamic, all these approaches simply assume a passive adversary, or only provide an informal/non-standard security analysis for an active adversary. As a result, some of these protocols [3, 30] have been found to be flawed in [26] and [10], respectively.

Research on provably-secure group key agreement in a formal security model is fairly new. It is only recently that Bresson et al. [15, 12, 13] have presented the first group key agreement protocols proven secure in a well-defined security model which extends earlier work of Bellare et al. [6, 8, 5] to the multi-party setting. The initial work [15] assumes that group membership is static, whereas later works [12, 13] focus on the dynamic case. But one drawback of their scheme is that in case of initial key agreement, its round complexity is linear in the number of group members. Moreover, the simultaneous joining of multiple users also takes a linear number of rounds with respect to the number of new members. Consequently, as the group size grows large, this scheme becomes impractical particularly in a wide area network environment where the delays associated with communication are expected to dominate the cost for group key agreement.

More recently, Katz and Yung [25] have proposed the first constant-round protocol for group key agreement that has been proven secure against an active adversary; the protocol requires three rounds of communication and achieves provable security under the Decisional Diffie-Hellman assumption in the standard model. Specifically, they provide a formal proof of security for the two-round protocol of Burmester and Desmedt [16], and introduce a one-round compiler that transforms any group key exchange protocol secure against a passive adversary into one that is secure against an active adversary with powerful capabilities. In this protocol all group members behave in a completely symmetric manner; in a group of size n, each member sends one broadcast message per round, and computes three modular exponentiations, $O(n \log n)$ modular multiplications, and $O(n)$ signature verifications. While the protocol is very efficient in general, this full symmetry negatively impacts the protocol performance in a scenario similar to our setting; the communication overhead is significant with three rounds of n broadcasts, and furthermore, the protocol has to restart from scratch in the presence of any membership change.

In [10] Boyd and Nieto have introduced a one-round group key agreement protocol which is provably secure in the random oracle model [7]. This protocol is computationally asymmetric and thus, as is the case with other asymmetric protocols [29, 24, 12, 13], appears to be easily extended to address the dynamic case. But unfortunately, this protocol does not achieve forward secrecy even if its round complexity is optimal. Thus it still remains an open problem to find a forward-secure group key exchange scheme running in a single round.

Most recently, Bresson and Catalano [11] have presented another provably-secure protocol

Table 1: Complexity comparison among group key agreement schemes that achieve both provable security and forward secrecy

		Communication				Computation	
		Rounds	Messages	Unicasts	Broadcasts	Exp.	Ver.
[12]	IKA	$n^{1)}$	n	$n-1$	1	$O(n^2)$	$O(n)$
	Join	$j+1$	$j+1$	$j^{2)}$	1	$O(jn)$	$O(n)$
	Leave	1	1		1	$O(n)$	$O(n)$
[25]		3	$3n$		$3n$	$O(n)+O(n^2\log n)^{3)}$	$O(n^2)$
Here	IKA	2	n	$n-1$	1	$O(n)^{4)}$	$O(n)$
	Join	2	$j+1$	j	1	$O(n)^{4)}$	$O(n)$
	Leave	1	1		1	$O(n)^{4)}$	$O(n)$

IKA: Initial Key Agreement, Exp: Modular Exponentiation, Ver: Signature Verification

1) The number of users in a newly updated group
2) The number of joining users
3) $O(n^2 \log n)$: the number of modular multiplications
4) The number of exponentiations in $\mathbb{G}$ defined in Section 2.1

which completes in two rounds of communication. Interestingly, unlike previous approaches, they construct the protocol by combining the properties of the ElGamal encryption scheme [19] with standard secret sharing techniques [28]. However, this protocol suffers from a significant communication overhead both in terms of the number of messages sent by all members during the protocol execution and in terms of the number of bits communicated throughout the protocol. Moreover, like the protocol of Katz and Yung [25], this protocol intends to exchange a session key in a scenario where the membership is static.

1.2 Our Contribution

The unsatisfactory situation described above has prompted this work aimed at designing an efficient and provably-secure key agreement scheme for a dynamic group where users communicate over a high-delay network environment. We provide a rigorous proof of security of the scheme in the model of Bresson et al. [15, 12, 13] in which an adversary controls all communication flows in the network. The concrete security reduction we exhibit in the ideal hash model is tight; breaking the semantic security of our scheme almost always leads to solving the well-established factoring problem, provided that the signature scheme used is existentially unforgeable. Our group key agreement scheme also provides perfect forward secrecy [18]; i.e., disclosure of long-term secret keys does not compromise the security of previously established session keys.

In wide area network environments, the main source of delay is not the computational time needed for cryptographic operations, but the communication time spent in the network.[1] Moreover, the power of computers continues to increase at a rapid pace. We refer the reader to the literature [2, 24] for detailed discussions of comparison between the communication latency in wide area networks and the computation time for modular exponentiation. As the experiment results of [2] also indicate, it is widely accepted that the number of communication rounds and the number of exchanged messages are two most important factors for efficient key agreement over a high-delay network.

Table 1 compares the efficiency of our scheme given in Section 5 with other provably-secure

[1] For example, the computation of a modular exponentiation $x^y \bmod z$ with $|x| = |y| = |z| = 1024$ takes about 9 ms using the big number library in OpenSSL on a Athlon XP 2100+ PC, whereas a 100-300 ms round-trip delay in wide area networks is common.

schemes that provide forward secrecy [12, 25]. As for computational costs, the table lists the total amount of computation that needs to be done by group members. As shown in the table, the scheme of [12] requires n communication rounds for initial key agreement which occurs at the time of group genesis, and j communication rounds for the rekeying operation that follows the joining of j new users. The protocol of [25], as already mentioned, requires n broadcast messages to be sent in each of three rounds, both for initial key agreement and for every group rekeying operation. In contrast, our scheme takes at most 2 communication rounds while maintaining low message complexity, in any of the three cases. Therefore, it is straightforward to see that our dynamic group key agreement scheme is well suited for networking environments with high communication latency. In particular, due to its computational asymmetry, our scheme is best suited for unbalanced networks consisting of mobile hosts with restricted computational resources and stationary hosts with relatively high computational capabilities.

The remainder of this paper is organized as follows. We begin with some notations and background in Section 2. We continue with a description of the standard security model for group key agreement protocols in Section 3. Then, in Section 4, we define the security of an authenticated key agreement protocol for a dynamic group, and describe the underlying assumptions on which the security of our scheme is based. Finally, we introduce a dynamic group key agreement scheme in Section 5 and give a security proof for this scheme in the random oracle model in Section 6.

2 Preliminaries

In this section we first set up some notations which will be used throughout this paper (even if some other notations are given locally near its first use). Then, as a preliminary step towards the security proof in Section 6, we describe some number theoretic properties of the finite cyclic group defined below.

2.1 Notations

Let N be the product of two large distinct primes p and q of equal length such that $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also prime integers. Then such an N is a Blum integer since $p \equiv q \equiv 3 \pmod 4$. We denote by $\mathbb{Z}_N^*$ the multiplicative group modulo N. An element $v \in \mathbb{Z}_N^*$ is called a quadratic residue modulo N if there exists an $x \in \mathbb{Z}_N^*$ such that $x^2 \equiv v \pmod N$. If no such x exists, then v is called a quadratic non-residue modulo N. We denote by $g \neq 1$ a quadratic residue that is chosen uniformly at random in the set of quadratic residues in $\mathbb{Z}_N^*$. Using this quadratic residue g, we define the finite group $\mathbb{G}$, over which we must work, to be $\mathbb{G} = \langle g \rangle$ where $\langle g \rangle$ is the cyclic subgroup of $\mathbb{Z}_N^*$ generated by g.

2.2 Background

Jacobi Symbol. The Jacobi symbol $(\frac{v}{N})$ of an element $v \in \mathbb{Z}_N^*$ is a polynomial time computable function which is defined as

$$(\frac{v}{N}) = (\frac{v}{p})(\frac{v}{q}),$$

where the symbols on the right are the Legendre symbols. However, the Jacobi symbol $(\frac{v}{N})$ can be efficiently computed even if the factorization of N is unknown, and moreover, it provides some information about the quadratic residuosity of v in $\mathbb{Z}_N^*$. If $(\frac{v}{N})$ is -1, then $(\frac{v}{p}) = -1$ or $(\frac{v}{q}) = -1$ and thus v is a quadratic non-residue modulo N. If v is a quadratic residue modulo N, then the Jacobi symbol $(\frac{v}{N})$ evaluates to 1. However, $(\frac{v}{N}) = 1$ does not imply that v is a quadratic residue modulo N. In summary, v is a quadratic residue modulo N only if $(\frac{v}{N})$ is 1, and $(\frac{v}{N})$ is -1 only if v is a quadratic non-residue modulo N.

Blum Integers. It is well known that a Blum integer $N = p \cdot q$ has the following properties.

- Among four square roots of each quadratic residue modulo N, there exists exactly one square root that is also a quadratic residue modulo N. In other words, squaring is a permutation over the set of quadratic residues in $\mathbb{Z}_N^*$. To see this, it is enough to note that $(\frac{-1}{p}) = -1$ and $(\frac{-1}{q}) = -1$, and for $v \in \mathbb{Z}_N^*$, v is a quadratic residue modulo N if and only if $(\frac{v}{p}) = 1$ and $(\frac{v}{q}) = 1$.

- For $u, v \in \mathbb{Z}_N^*$, let $(\frac{u}{N}) = 1$ and $(\frac{v}{N}) = -1$, and let $u^2 \equiv v^2 \pmod N$. Then $u \neq \pm v \bmod N$ and therefore $\Pr[\gcd(u - v, N) \in \{p, q\}] = 1$. To see this, it suffices to observe that $(\frac{-1}{N}) = 1$.

Quadratic Residues. We now describe some properties of quadratic residues in $\mathbb{Z}_N^*$ observed in the work of Biham *et al.* [9]. Let QR_N denote the set of quadratic residues in $\mathbb{Z}_N^*$. Then the cardinality of QR_N is odd which is evident from

$$|QR_N| = \varphi(N)/4 = (p-1) \cdot (q-1)/4 = p'q', \tag{1}$$

where $\varphi(\cdot)$ denotes the Euler Phi function.

From Equation (1) and since QR_N forms a multiplicative subgroup of $\mathbb{Z}_N^*$, it follows that the order of any quadratic residue $\alpha \in \mathbb{Z}_N^*$ is odd (i.e., 1, p', q', or $p'q'$). Then, because 2 is relatively prime to $m = |\mathbb{G}| = |\langle g \rangle|$ (i.e., $\gcd(2, m) = 1$), we know that $2 \in \mathbb{Z}_m^*$. Namely, $2^{-1} \bmod m$ exists and is nothing but $(m + 1)/2$. Therefore, $g^{2^{-1} \bmod m} \bmod N$ is equal to $g^{(m+1)/2} \bmod N$ which is not only a quadratic residue modulo N, but also a square root of g. Similarly, $g^{2^{-2} \bmod m} \bmod N = g^{((m+1)/2)^2 \bmod m} \bmod N$ is the unique square root of $g^{2^{-1} \bmod m} \bmod N$ that is a quadratic residue modulo N.

3 The Model

Since the work of Bresson et al. [15], the formal security model described here has been widely used in the literature [12, 13, 14, 25, 10] to properly analyze the security of group key agreement schemes. In this work we slightly modify the model of Bresson et al. [12] which is the first formal security model that explicitly deals with the dynamic case.

Participants. Let $\mathcal{U} = \{U_1, U_2, \ldots, U_{p_u(k)}\}$ be the universe of all users that can participate in a group key agreement scheme, where $p_u(k)$ is a polynomial function of the security parameter k. Let $\mathcal{MG}$ be a subset of $\mathcal{U}$ called a multicast group, the users of which wish to establish a session key among them. Then, in $\mathcal{MG}$, one user plays a special role which will be made clear in the description of the scheme in Section 5. We call this user the *controller* and each of the other users in $\mathcal{MG}$ a *non-controller*. The role of each user as either a controller or a non-controller is assigned by the adversary $\mathcal{A}$, as shown later in this section.

In initialization phase each user U_i in $\mathcal{U}$ obtains a long-term public/private key pair (PK_i, SK_i) by running a key generation algorithm $\mathcal{G}(1^k)$. The set of public keys of all users is assumed to be known a priori to all parties including the adversary $\mathcal{A}$.

Partnering. Intuitively, the *partner ID* for any user is the set of all the users that should compute the same session key as that user in a protocol execution. The partner ID is defined via the *session ID* which in turn is defined as a function of the messages exchanged among users in that protocol execution.

Before we define partnering among users, we first need to describe the basic structure of our scheme. The scheme consists of three protocols IKA1, LP1, and JP1 for initial group formation, user leave, and user join, respectively. In each protocol participants are one controller and one or more non-controllers; the controller exchanges messages with all other non-controllers whereas a non-controller exchanges messages only with the controller. In a protocol execution a user is said to *accept* when it has computed a session key as per protocol specification.

With the above in mind, we now define the session ID for each user U_i which is denoted by SID_i. The session ID for a user is initially set to $\emptyset$ in a protocol execution and is defined when the user accepts in that execution. Specifically, if U_i is a non-controller, then SID_i is defined as $\mathsf{SID}_i = \{M\}$, where M is the concatenation of all messages sent and received by U_i. If instead U_i is the controller, we define SID_i as

$$\mathsf{SID}_i = \{M_i^j \mid U_j \in \mathcal{P}_i\},$$

where $\mathcal{P}_i$ is the set of all users with which user U_i has exchanged some messages, and M_i^j is the concatenation of all messages that U_i has exchanged with U_j.

Using the session ID defined above, we now define the partner ID for user U_i which is denoted by PID_i. Let ACC_i be a variable that is `TRUE` if U_i has accepted, and `FALSE` otherwise. Then we define the partner ID for U_i to be

$$\begin{aligned}\mathsf{PID}_i = \{U_j \mid\ & \mathsf{SID}_i \cap \mathsf{SID}_k \neq \emptyset \wedge \mathsf{SID}_k \cap \mathsf{SID}_j \neq \emptyset \wedge \\ & \mathsf{ACC}_i = \mathsf{ACC}_k = \mathsf{ACC}_j = \texttt{TRUE},\ \text{for some } U_k\}.\end{aligned}$$

Note that in the above definition of partner ID, it is possible that $U_i = U_k$. Therefore, the conjunction simply says that user U_j is a *partner* of user U_i if $\mathsf{SID}_i \cap \mathsf{SID}_j \neq \emptyset$ and $\mathsf{ACC}_i = \mathsf{ACC}_j = \texttt{TRUE}$, or they share the same partner U_k. All SIDs and PIDs are public and hence available to the adversary $\mathcal{A}$.

Adversary. Along with a set of protocol participants, the model also includes the adversary $\mathcal{A}$ who controls all communication flows in the network. The adversary interacts with users through various queries, each of which captures a capability of the adversary. Listed below are the queries which are allowed for adversary $\mathcal{A}$ to make.

- $\mathsf{Send}(U_i, m)$: This query models the ability of adversary $\mathcal{A}$ sending a message m to a user U_i. Upon receiving the message m, user U_i is assumed to proceed as specified in the protocol in which it is participating; the user updates its state and sends out a response message as needed. The response message is returned to adversary $\mathcal{A}$. Queries of the form $\mathsf{Send}(U_i, m1 : \mathcal{MG} : m2)$, where $m1 \in \{$"`IKA1`", "`LP1`", "`JP1`"$\}$ and $m2 \in \{$"`controller`", "`non-controller`"$\}$, allow adversary $\mathcal{A}$ to initiate a protocol execution among the users in $\mathcal{MG}$, specifying the role of U_i in this execution.

- $\mathsf{Reveal}(U_i)$: This query models the misuse of the session key by the users. If U_i has accepted holding a session key K, then the query divulges K to adversary $\mathcal{A}$.

- $\mathsf{Corrupt}(U_i)$: This query outputs the long-term private key SK_i of user U_i.

- $\mathsf{Test}(U_i)$: This query models the semantic security of the session key K and is answered as follows: one flips a secret coin b, and returns the real session key K if $b = 1$ or else a random string chosen from $\{0,1\}^\ell$ if $b = 0$, where ℓ is the length of the session key to be distributed in the protocol. This query can be made at most once, only to a *fresh* user (see below for the definition of "fresh user").

Freshness. As mentioned above, the query $\mathsf{Test}(U_i)$ can be asked only when user U_i is fresh. We say that a user U_i is *fresh* in the current protocol execution if all the following conditions hold: (1) $\mathsf{ACC}_i = \mathtt{TRUE}$, (2) no one in PID_i has been asked for a Reveal query (note that $U_i \in \mathsf{PID}_i$ unless $\mathsf{PID}_i \neq \emptyset$), and (3) no one in $\mathcal{U}$ has ever been asked for a $\mathsf{Corrupt}$ query since the initialization phase.

4 Security Definitions

In this section we first define what it means to securely distribute a session key within the security model given above and then explore the underlying assumptions on which the security of our scheme rests.

Authenticated Group Key Agreement. The security of an authenticated group key agreement scheme P is defined in the following context. The adversary $\mathcal{A}$, equipped with all the queries described in the security model, executes the protocols IKA1, LP1, and JP1 as many times as she wishes in an arbitrary order, of course, with IKA1 being the first one executed. During executions of the protocols, the adversary $\mathcal{A}$, at any time, asks a Test query to a fresh user, gets back an ℓ-bit string as the response to this query, and at some later point in time, outputs a bit b' as a guess for the secret bit b. Let GG ($\mathsf{Good\ Guess}$) be the event that the adversary $\mathcal{A}$ correctly guesses the bit b, i.e., the event that $b' = b$. Then we define the advantage of $\mathcal{A}$ in attacking P as

$$\mathsf{Adv}_P^{\mathcal{A}}(k) = 2 \cdot \Pr[\mathsf{GG}] - 1.$$

We say that a group key agreement scheme P is secure if $\mathsf{Adv}_P^{\mathcal{A}}(k)$ is negligible for any probabilistic polynomial time adversary $\mathcal{A}$.

Secure Signature Schemes. We review here the standard definition of a digital signature scheme. A digital signature scheme $\Gamma = (\mathcal{G}, \mathcal{S}, \mathcal{V})$ is defined by the following triple of algorithms:

- A *probabilistic key generation algorithm* $\mathcal{G}$, on input 1^k, outputs a pair of matching public and private keys (PK, SK).
- A *signing algorithm* $\mathcal{S}$ is a (possibly probabilistic) polynomial time algorithm that, given a message m and a key pair (PK, SK) as inputs, outputs a signature σ of m.
- A *verification algorithm* $\mathcal{V}$ is a (usually deterministic) polynomial time algorithm that on input (m, σ, PK), outputs 1 if σ is a valid signature of the message m with respect to PK, and 0 otherwise.

We denote by $\mathsf{Succ}_\Gamma^{\mathcal{A}}(k)$ the probability of an adversary $\mathcal{A}$ succeeding with an existential forgery under adaptive chosen message attack [20]. We say that a signature scheme Γ is secure if $\mathsf{Succ}_\Gamma^{\mathcal{A}}(k)$ is negligible for any probabilistic polynomial time adversary $\mathcal{A}$. We denote by $\mathsf{Succ}_\Gamma(t)$ the maximum value of $\mathsf{Succ}_\Gamma^{\mathcal{A}}(k)$ over all adversaries $\mathcal{A}$ running in time at most t.

Factoring Assumption. Let $\mathcal{FIG}$ be a factoring instance generator that on input 1^k, runs in time polynomial in k and outputs a $2k$-bit integer $N = p \cdot q$, where p and q are as defined in Section 2.1. Then, we define $\mathsf{Succ}_N^{\mathcal{A}}(k)$ as the advantage of adversary $\mathcal{A}$ in factoring $N = p \cdot q$ chosen from $\mathcal{FIG}(1^k)$. Namely,

$$\mathsf{Succ}_N^{\mathcal{A}}(k) = \Pr[\mathcal{A}(N) \in \{p, q\} \mid N(=pq) \longleftarrow \mathcal{FIG}(1^k)].$$

We say that $\mathcal{FIG}$ satisfies the factoring assumption if for all sufficiently large k, $\mathsf{Succ}_N^{\mathcal{A}}(k)$ is negligible for any probabilistic polynomial time adversary $\mathcal{A}$. Similarly as before, we denote by $\mathsf{Succ}_N(t)$ the maximum value of $\mathsf{Succ}_N^{\mathcal{A}}(k)$ over all adversaries $\mathcal{A}$ running in time at most t.

5 The Proposed Scheme

We now present a dynamic group key agreement scheme consisting of three protocols IKA1, LP1, and JP1 for initial group formation, user leave, and user join, respectively.

Let N be any possible output of $\mathcal{FIG}(1^k)$, and let $g \neq 1$ and $\mathbb{G}$ be as defined in Section 2.1. For the rest of the paper, we denote by U_c the controller in a multicast group $\mathcal{MG}$, and by $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\ell$ a hash function modelled as a random oracle in the security proof of the scheme. For simplicity, we will often omit "mod N" from expressions if no confusion arises.

5.1 Initial Key Agreement: Protocol IKA1

Assume a multicast group $\mathcal{MG} = \{U_1, U_2, \ldots, U_n\}$ of n users who wish to establish a session key by participating in protocol IKA1. Then IKA1 runs in two rounds, one with $n-1$ unicasts and the other with a single broadcast, as follows:

1. Each U_i picks a random $r_i \in [1, N]$ and computes $z_i = g^{r_i} \bmod N$. $U_i \neq U_c$ then signs $U_i \| z_i$ to obtain signature σ_i and sends $m_i = U_i \| z_i \| \sigma_i$ to the controller U_c.

2. Upon receiving each message m_i, U_c verifies the correctness of m_i and computes $y_i = z_i^{r_c} \bmod N$. After receiving all the $n-1$ messages, U_c computes Y as $Y = \prod_{i \in [1,n] \setminus \{c\}} y_i$ mod$N$ if n is even, and as $Y = \prod_{i \in [1,n]} y_i \bmod N$ if n is odd. U_c also computes the set $\mathcal{T} = \{T_i \mid i \in [1,n] \setminus \{c\}\}$ where $T_i = Y \cdot y_i^{-1} \bmod N$. Let $\mathcal{Z} = \{z_i \mid i \in [1,n]\}$. Then, U_c signs $\mathcal{MG} \| \mathcal{Z} \| \mathcal{T}$ to obtain signature σ_c and broadcasts $m_c = \mathcal{MG} \| \mathcal{Z} \| \mathcal{T} \| \sigma_c$ to the entire group.

3. Upon receiving the broadcast message m_c, each $U_i \neq U_c$ verifies the correctness of m_c and computes $Y = z_c^{r_i} \cdot T_i \bmod N$. All users in $\mathcal{MG}$ compute their session key as $K = \mathcal{H}(\mathcal{T} \| Y)$, and store their random exponent r_i and the set $\mathcal{Z}$ for future use.

To take a simplified example as an illustration, consider a multicast group $\mathcal{MG} = \{U_1, U_2, \ldots, U_5\}$ and let $U_c = U_5$. Then, in IKA1, the controller U_5 receives $\{g^{r_1}, g^{r_2}, g^{r_3}, g^{r_4}\}$ from the rest of the users, and broadcasts $\mathcal{Z} = \{g^{r_1}, g^{r_2}, g^{r_3}, g^{r_4}, g^{r_5}\}$ and $\mathcal{T} = \{g^{r_5(r_2+r_3+r_4+r_5)}, g^{r_5(r_1+r_3+r_4+r_5)}, g^{r_5(r_1+r_2+r_4+r_5)}, g^{r_5(r_1+r_2+r_3+r_5)}\}$. All users in $\mathcal{MG}$ compute the same key: $K = \mathcal{H}(\mathcal{T} \| Y)$, where $Y = g^{r_5(r_1+r_2+r_3+r_4+r_5)}$.

5.2 User Leave: Protocol LP1

Assume a scenario where a set of users $\mathcal{L}$ leaves a multicast group $\mathcal{MG}_p$. Then protocol LP1 is executed to provide each user of the new multicast group $\mathcal{MG}_n = \mathcal{MG}_p \setminus \mathcal{L}$ with a new session key. Any remaining user can act as the controller in the new multicast group $\mathcal{MG}_n$. LP1 requires only one communication round with a single broadcast and it proceeds as follows:

1. U_c picks a new random $r'_c \in [1, N]$ and computes $z'_c = g^{r'_c} \bmod N$. Using r'_c, z'_c and the saved set $\mathcal{Z}$, U_c then proceeds exactly as in IKA1, except that it broadcasts $m_c = \mathcal{MG}_n \| z_c \| z'_c \| \mathcal{T} \| \sigma_c$ where z_c is the random exponential from the previous controller.

2. Upon receiving the broadcast message m_c, each $U_i \neq U_c$ verifies that: (1) $\mathcal{V}(\mathcal{MG}_n \| z_c \| z'_c \| \mathcal{T}, \sigma_c, PK_c) = 1$ and (2) the received z_c is equal to the one that is received in the previous session. All users in $\mathcal{MG}_n$ then compute their session key as $K = \mathcal{H}(\mathcal{T} \| Y)$ and update the set $\mathcal{Z}$.

We assume that in the previous example, a set of users $\mathcal{L} = \{U_2, U_4\}$ leaves the multicast group $\mathcal{MG}_p = \{U_1, U_2, \ldots, U_5\}$ and hence the remaining users form a new multicast group $\mathcal{MG}_n = \{U_1, U_3, U_5\}$. Also assume that U_5 remains as the controller in the new multicast group $\mathcal{MG}_n$. Then U_5 chooses a new random value r'_5, and broadcasts z_5, $z'_5 = g^{r'_5}$, and $\mathcal{T} = \{g^{r'_5(r_3+r'_5)}, g^{r'_5(r_1+r'_5)}\}$. All users in $\mathcal{MG}_n$ compute the same key: $K = \mathcal{H}(\mathcal{T} \| Y)$, where $Y = g^{r'_5(r_1+r_3+r'_5)}$.

5.3 User Join: Protocol JP1

Assume a scenario in which a set of j new users, $\mathcal{J}$, joins a multicast group $\mathcal{MG}_p$ to form a new multicast group $\mathcal{MG}_n = \mathcal{MG}_p \cup \mathcal{J}$. Then the join protocol JP1 is run to provide the users of $\mathcal{MG}_n$ with a session key. Any user from the previous multicast group $\mathcal{MG}_p$ can act as the controller in the new multicast group $\mathcal{MG}_n$. JP1 takes two communication rounds, one with j unicasts and the other with a single broadcast, and it proceeds as follows:

1. Each $U_i \in \mathcal{J}$ picks a random $r_i \in [1, N]$ and computes $z_i = g^{r_i} \bmod N$. $U_i \in \mathcal{J}$ then generates signature σ_i of $U_i \| z_i$, sends $m_i = U_i \| z_i \| \sigma_i$ to U_c, and stores its random r_i.

2. U_c proceeds in the usual way, choosing a new random r'_c, computing z'_c, Y, $\mathcal{T}$ and $K = \mathcal{H}(\mathcal{T} \| Y)$, updating the set $\mathcal{Z}$ with new z_i's, and then broadcasting $m_c = \mathcal{MG}_n \| z_c \| \mathcal{Z} \| \mathcal{T} \| \sigma_c$.

3. After verifying the correctness of m_c (including the verification by $U_i \in \mathcal{MG}_p \setminus \{U_c\}$ that the received z_c is equal to the one received in the previous session), each $U_i \neq U_c$ proceeds as usual, computing $Y = z'^{r_i}_c \cdot T_i \bmod N$ and $K = \mathcal{H}(\mathcal{T} \| Y)$. All users in $\mathcal{MG}_n$ store or update the set $\mathcal{Z}$.

Consider the same example as used for LP1 and assume that a set of users $\mathcal{J} = \{U_2\}$ joins the multicast group $\mathcal{MG}_p = \{U_1, U_3, U_5\}$ to form a new multicast group $\mathcal{MG}_n = \{U_1, U_2, U_3, U_5\}$. Also assume that controller $U_c = U_5$ remains unchanged from $\mathcal{MG}_p$ to $\mathcal{MG}_n$. Then, U_5 receives $\{g^{r'_2}\}$ from the users in $\mathcal{J}$, and broadcasts z'_5, $\mathcal{Z} = \{g^{r_1}, g^{r'_2}, g^{r_3}, g^{r''_5}\}$ and $\mathcal{T} = \{g^{r''_5(r'_2+r_3)}, g^{r''_5(r_1+r_3)}, g^{r''_5(r_1+r'_2)}\}$ to the rest of the users, where r''_5 is the new random exponent of controller U_5. All users in $\mathcal{MG}_n$ compute the same key: $K = \mathcal{H}(\mathcal{T} \| Y)$, where $Y = g^{r''_5(r_1+r'_2+r_3)}$.

6 Security Analysis

Theorem 1. *Let $\mathsf{Adv}_P(t, q_{se}, q_h)$ be the maximum advantage in attacking P, where the maximum is over all adversaries that run in time t, and make q_{se} Send queries and q_h random oracle queries. Then we have*

$$\mathsf{Adv}_P(t, q_{se}, q_h) \leq 2 \cdot \mathsf{Succ}_N(t') + 2p_u(k) \cdot \mathsf{Succ}_\Gamma(t''),$$

where $t' = t + O(q_{se}p_u(k)t_{exp} + q_h t_{exp})$, $t'' = t + O(q_{se}p_u(k)t_{exp})$, and t_{exp} is the time required to compute a modular exponentiation in $\mathbb{G}$.

In the following we briefly outline the proof of Theorem 1. The proof is divided into two cases: (1) the case that the adversary $\mathcal{A}$ breaks the scheme by forging a signature with respect to some user's public key, and (2) the case that $\mathcal{A}$ breaks the scheme without forging a signature. We argue by contradiction, assuming that there exists an adversary $\mathcal{A}$ who has a non-negligible advantage in attacking P. For the case (1), we reduce the security of scheme P to the security of the signature scheme Γ, by constructing an efficient forger $\mathcal{F}$ who given as input a public key PK and access to a signing oracle associated with this key, outputs a valid forgery with respect to PK. For the case (2), the reduction is from the factoring problem; given the adversary $\mathcal{A}$, we build an efficient factoring algorithm $\mathcal{B}$ which given as input $N = p \cdot q$ generated by $\mathcal{FIG}(1^k)$, outputs either p or q.

Proof. Assume by contradiction that there exists an adversary $\mathcal{A}$ who has a non-negligible advantage in attacking the scheme P. Then we will show that either an efficient signature forger $\mathcal{F}$ against Γ or an efficient factoring algorithm $\mathcal{B}$ for N can be constructed from the adversary $\mathcal{A}$.

6.1 Signature Forger $\mathcal{F}$

Assume that the adversary $\mathcal{A}$ gains its advantage by forging a signature with respect to some user's public key. Then we build from $\mathcal{A}$ a signature forger $\mathcal{F}$ against the signature scheme Γ. The forger $\mathcal{F}$, given as input a public key PK and access to a signing oracle associated with this key, outputs a valid forgery (m, σ) with respect to PK, i.e., $\mathcal{V}(m, \sigma, PK) = 1$ such that σ was not previously output by the signing oracle as a signature on the message m.

$\mathcal{F}$ begins by choosing at random a user $U_f \in \mathcal{U}$, and setting PK_f to PK. For all other users, $\mathcal{F}$ honestly generates a public/private key pair by running the key generation algorithm $\mathcal{G}(1^k)$. $\mathcal{F}$ then invokes $\mathcal{A}$ and simulates the queries from $\mathcal{A}$ as follows:

- $\mathsf{Send}(U_i, m)$: If $i \neq f$, $\mathcal{F}$ knows the private signing key of U_i, and hence can answer the queries following the scheme exactly as specified. If instead $i = f$, then $\mathcal{F}$ does not have the private signing key of U_i. Nevertheless, $\mathcal{F}$ can obtain signatures of any messages it wants by accessing the signing oracle associated with PK.
- $\mathsf{Reveal}(U_i)$ / $\mathsf{Test}(U_i)$: These queries are answered in the obvious way.
- $\mathsf{Corrupt}(U_i)$: If $U_i \neq U_f$, then $\mathcal{F}$ simply hands the private key SK_i which was generated by $\mathcal{F}$ itself. However, if $\mathcal{A}$ corrupts $U_i = U_f$, then $\mathcal{F}$ does not have the associated private key, and so halts and outputs "`fail`".

The simulation provided above is perfectly indistinguishable from the real execution unless adversary $\mathcal{A}$ makes the query $\mathsf{Corrupt}(U_f)$. Throughout this simulation, $\mathcal{F}$ monitors each Send query from $\mathcal{A}$, and checks if it includes a valid message/signature pair (m, σ) with respect to PK. If no such query is made until $\mathcal{A}$ stops, then $\mathcal{F}$ halts and outputs "`fail`". Otherwise, $\mathcal{F}$ outputs (m, σ) as a valid forgery with respect to PK.

Now, we quantify the success probability of $\mathcal{F}$ in outputting a forgery in the simulation above. Let Forge be the event that $\mathcal{A}$ outputs a valid forgery with respect to the public key PK_i of some user $U_i \in \mathcal{U}$ before making the query $\mathsf{Corrupt}(U_i)$. Then, since $\mathsf{Succ}_\Gamma^{\mathcal{F}}(k) = \Pr[\mathsf{Forge}]/p_u(k)$, it follows by definition that

$$\Pr[\mathsf{Forge}] \leq p_u(k) \cdot \mathsf{Succ}_\Gamma(t''). \tag{2}$$

In the simulation above, $\mathcal{F}$ performs at most $p_u(k)$ modular exponentiations to answer a Send query, and all other queries (Reveal, $\mathsf{Corrupt}$ or Test) can be trivially answered. Therefore, since the running time of $\mathcal{F}$ is the running time of $\mathcal{A}$ plus the time required to process all the queries from $\mathcal{A}$, we have $t'' = t + O(q_{se} p_u(k) t_{exp})$ as claimed.

6.2 Factoring Algorithm $\mathcal{B}$

The basic idea of the proof given here is inspired by the technique of Biham, et al. [9], where they showed that breaking the generalized Diffie-Hellman assumption modulo a Blum integer is at least as hard as factoring Blum integers.

Assume that the adversary $\mathcal{A}$ breaks the scheme P without forging a signature. Then, we construct from $\mathcal{A}$ an efficient factoring algorithm $\mathcal{B}$ which given as input a Blum integer $N = p \cdot q$ chosen from $\mathcal{FIG}(1^k)$, outputs either p or q. $\mathcal{B}$ begins by running $\mathcal{G}(1^k)$ to generate (PK_i, SK_i) for each user $U_i \in \mathcal{U}$, and setting $g = v^{2^2} \bmod N$ where v is an integer chosen uniformly at random in $\mathbb{Z}_N^*$ such that the Jacobi symbol $(\frac{v}{N})$ is -1. Because N is a Blum integer, v^2 is a uniformly distributed quadratic residue in $\mathbb{Z}_N^*$ and furthermore, squaring is a permutation on the set of quadratic residues in $\mathbb{Z}_N^*$. Therefore, g is also a uniformly distributed quadratic residue in $\mathbb{Z}_N^*$. Let d be the order of g in $\mathbb{Z}_N^*$, which of course is unknown to $\mathcal{B}$. Then, since d is always odd, we have that $2 \in \mathbb{Z}_d^*$; i.e., $2^{-1} \bmod d$ exists. For brevity, we use $g^{2^{-i}} \bmod N$ to denote $g^{2^{-i} \bmod d} \bmod N$ for $i = 1, 2$. $\mathcal{B}$ now invokes $\mathcal{A}$ and simulates all the queries from $\mathcal{A}$ as follows.

- Send: $\mathcal{B}$ handles all the Send queries of $\mathcal{A}$ as per the specifications of the protocols, except that it computes each z_i in the following different yet indistinguishable way. $\mathcal{B}$ first selects a random $a_i \in [1, N]$ and then computes z_i as

  $$z_i = g^{r_i} = g^{a_i + 2^{-1}} = g^{a_i} \cdot g^{2^{-1}} = g^{a_i} \cdot v^2,$$

  where the computations are all mod N. Notice that the random exponent r_i denotes the value $a_i + 2^{-1} \bmod d$ which, of course, is unknown to $\mathcal{B}$. $\mathcal{B}$ records the tuple $\langle z_i, a_i \rangle$ for its own use.

  We now show that $\mathcal{B}$ can correctly compute the set $\mathcal{T}$ even if it does not know any of the random exponents. Without loss of generality, let $\mathcal{MG} = \{U_1, U_2, \ldots, U_n\}$ be the multicast group of n users who are participating in the current protocol execution and assume that $\mathcal{B}$ has obtained all the tuples $\langle z_i, a_i \rangle$ for $i \in [1, n]$. Then if n is odd, $\mathcal{B}$ computes each T_i as follows:

  $$\begin{aligned} T_i &= \prod_{j \in [1,n] \setminus \{i\}} z_j^{r_c} \\ &= g^{r_c \cdot \sum_{j \in [1,n] \setminus \{i\}} r_j} \\ &= (g^{r_c})^{(n-1)/2 + \sum_{j \in [1,n] \setminus \{i\}} a_j} \\ &= z_c^{(n-1)/2} \cdot \prod_{j \in [1,n] \setminus \{i\}} z_c^{a_j}, \end{aligned}$$

  where the computations are all mod N. The equation for the case of even n requires only a minor modification to the equation above, and we omit it here.

- $\mathcal{H}$/Reveal: $\mathcal{B}$ simulates the random oracle $\mathcal{H}$ by assigning a random string h_δ from $\{0,1\}^\ell$ to each fresh query δ, and then adding the tuple $\langle \delta, h_\delta \rangle$ to the random oracle simulation list $\mathcal{HL}$. If the query δ is not new, then the answer is retrieved from the list $\mathcal{HL}$.

  We now describe how to answer Reveal queries. As can be seen from the way $\mathcal{B}$ handles the Send queries from $\mathcal{A}$, there is no session key available to $\mathcal{B}$ in this simulation. However, all Reveal queries can be simulated by using the fact that the session keys distributed in the scheme are outputs of random oracle $\mathcal{H}$. To aid the simulation, $\mathcal{B}$ maintains a special list $\mathcal{RL}$ which contains information related to all the revealed (fake) session keys. To be concrete, suppose that $\mathcal{A}$ has made the query $\mathsf{Reveal}(U_i)$ when no one in PID_i has been asked for a Reveal query. Then $\mathcal{B}$ selects a random string $h_{\mathcal{T}}$ from $\{0,1\}^\ell$ to represent the genuine session key $\mathcal{H}(\mathcal{T} \parallel Y)$, answers the query $\mathsf{Reveal}(U_i)$ with $h_{\mathcal{T}}$, and adds the tuple $\langle \mathcal{T}, h_{\mathcal{T}} \rangle$ to the list $\mathcal{RL}$. If instead some user in PID_i has been revealed before the query $\mathsf{Reveal}(U_i)$ is made, then $\mathcal{RL}$ must contain a tuple $\langle \mathcal{T}, h_{\mathcal{T}} \rangle$. In this case $\mathcal{B}$ simply returns the random string $h_{\mathcal{T}}$ taken from the list $\mathcal{RL}$.

There remains one thing to consider before we proceed to describe how to simulate other queries. Observe that $\mathcal{H}$ may have been queried on $\mathcal{T} \parallel Y$ at some time before the query $\mathsf{Reveal}(U_i)$ is made, or vice versa. This means that there is a possibility of inconsistency between answers of Reveal queries and random oracle queries. In other words, to represent the same value $\mathcal{H}(\mathcal{T} \parallel Y)$, $\mathcal{B}$ could end up using two different values: one as the answer to the random oracle query $\mathcal{T} \parallel Y$ and the other as the answer to the query $\mathsf{Reveal}(U_i)$. The main difficulty in providing the solution for this potential problem is the fact that the value Y is unknown to $\mathcal{B}$. But fortunately, we can circumvent this difficulty by using the following observation. Assume again a multicast group $\mathcal{MG} = \{U_1, U_2, \ldots, U_n\}$ of n users. Then, since for some $i \in [1, n] \setminus \{c\}$,

$$\begin{aligned} Y &\equiv (z_c)^{r_i} \cdot T_i \equiv g^{r_c \cdot r_i} \cdot T_i \\ &\equiv g^{(a_c + 2^{-1}) \cdot (a_i + 2^{-1})} \cdot T_i \\ &\equiv g^{a_c \cdot a_i + 2^{-1} \cdot (a_c + a_i) + 2^{-2}} \cdot T_i \pmod{N}, \end{aligned}$$

it is immediate that

$$\begin{aligned} g^{2^{-2}} &\equiv Y \cdot (g^{a_c \cdot a_i} \cdot g^{2^{-1}(a_c + a_i)} \cdot T_i)^{-1} \\ &\equiv Y \cdot (g^{a_c \cdot a_i} \cdot (v^2)^{a_c + a_i} \cdot T_i)^{-1} \pmod{N}. \end{aligned} \tag{3}$$

From (3) and since $(\frac{g^{2^{-2}} \bmod N}{N}) = 1$, it follows that given a value $Y' \in \mathbb{Z}_N^*$, the unknown value Y is equal to Y' *only if*

$$u^2 \equiv v^2 \pmod{N} \quad \text{and} \quad (\frac{u}{N}) = 1, \tag{4}$$

where $u = Y' \cdot (g^{a_i \cdot a_c} \cdot (v^2)^{a_i + a_c} \cdot T_i)^{-1} \bmod N$. Put succinctly, if $(\frac{u}{N}) = -1$ or u^2 is not congruent to $v^2 \bmod N$, then $Y \neq Y'$. Otherwise, since $(\frac{v}{N}) = -1$ and N is a Blum integer, it must be the case that $u \neq \pm v \bmod N$ and thus $\Pr[\gcd(u - v, N) \in \{p, q\}] = 1$.

This implies that $\mathcal{B}$ remains always able to answer correctly all the random oracle queries and Reveal queries of $\mathcal{A}$ as follows. Suppose that the query $\mathsf{Reveal}(U_i)$ is made by $\mathcal{A}$. If no one in PID_i has been asked for a Reveal query, then $\mathcal{B}$ searches all tuples in $\mathcal{HL}$ such that $\delta = \mathcal{T} \parallel Y'$ for some $Y' \in \mathbb{Z}_N^*$. For each such a tuple $\langle \delta, h_\delta \rangle$, $\mathcal{B}$ can either factor N or conclude $Y \neq Y'$, by using Equation (4). In the former case, $\mathcal{B}$ halts all the simulations and outputs $\gcd(u - v, N)$ as the final outcome. In the latter case, $\mathcal{B}$ proceeds to answer the query in the usual way, i.e., by returning a random string $h_{\mathcal{T}}$ from $\{0,1\}^\ell$ and adding the tuple $\langle \mathcal{T}, h_{\mathcal{T}} \rangle$ to $\mathcal{RL}$.

The case that the adversary $\mathcal{A}$ makes the random oracle query δ of the form $\mathcal{T} \parallel Y'$ can be worked out in an analogous way.

- $\mathsf{Corrupt}$: These queries are answered in the obvious way.
- Test: $\mathcal{B}$ simply returns a random string chosen from $\{0,1\}^\ell$.

Now, given the simulation above, let's consider the success probability of $\mathcal{B}$ in factoring N. Without loss of generality, we assume that $\mathcal{A}$ has made the Test query to a user whose unknown (real) session key is $\mathcal{H}(\mathcal{T}_{te} \parallel Y_{te})$. Let Ask be the event that $\mathcal{A}$ makes a query to $\mathcal{H}$ on $\mathcal{T}_{te} \parallel Y_{te}$. At some point, when $\mathcal{A}$ terminates and outputs its guess b', $\mathcal{B}$ simply checks the list $\mathcal{HL}$ to see if the event Ask has occurred, using the same way as it did for Reveal queries. If so, then $\mathcal{B}$ succeeds in factoring N. This is true because we are assuming here the case that $\mathcal{A}$ gains its advantage without forging a signature. Therefore, we have

$$\mathsf{Succ}_N^{\mathcal{B}}(k) \geq \Pr[\overline{\mathsf{Forge}} \wedge \mathsf{Ask}], \tag{5}$$

where the inequality is due to the possibility that $\mathcal{B}$ can succeed in factoring while answering Reveal queries or random oracle queries. Furthermore, since $\mathcal{A}$ cannot gain any advantage in guessing the bit b without making a query to $\mathcal{H}$ on $\mathcal{T}_{te} \parallel Y_{te}$, we obtain that $\Pr[\mathsf{GG} \mid \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ask}}] = 1/2$ and thus $\Pr[\mathsf{GG} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ask}}] \leq 1/2$. Now, from the assumption that the advantage of $\mathcal{A}$ in breaking P without forging a signature is non-negligible, it must be the case that $\Pr[\overline{\mathsf{Forge}} \wedge \mathsf{Ask}]$ is non-negligible. But then, by (5), this leads to the contradiction that there exists an factoring algorithm $\mathcal{B}$ whose success probability in factoring N is non-negligible. Therefore, we arrive at the conclusion that the advantage of $\mathcal{A}$ in breaking P without forging a signature is negligible.

Regarding the running time of $\mathcal{B}$, we see, as before, that processing the Send queries from $\mathcal{A}$ takes $O(q_{se}p_u(k)t_{exp})$. In addition, the amount of time required to process random oracle queries and Reveal queries is bounded by $O(q_h t_{exp})$. Hence we have that $t' = t + O(q_{se}p_u(k)t_{exp} + q_h t_{exp})$, since the running time of $\mathcal{B}$ is the running time of $\mathcal{A}$ added to the time needed to process all the queries from $\mathcal{A}$.

Now, it remains to quantify the advantage of $\mathcal{A}$ in attacking our scheme. A straightforward probability calculation shows that:

$$\begin{aligned}
\mathsf{Adv}_P^{\mathcal{A}}(k) &= 2 \cdot \Pr[\mathsf{GG}] - 1 \\
&= 2 \cdot \Pr[\mathsf{GG} \wedge \mathsf{Forge}] + 2 \cdot \Pr[\mathsf{GG} \wedge \overline{\mathsf{Forge}}] - 1 \\
&\leq 2 \cdot \Pr[\mathsf{Forge}] + 2 \cdot \Pr[\mathsf{GG} \wedge \overline{\mathsf{Forge}}] - 1 \\
&= 2 \cdot \Pr[\mathsf{Forge}] + 2(\Pr[\mathsf{GG} \wedge \overline{\mathsf{Forge}} \wedge \mathsf{Ask}] + \Pr[\mathsf{GG} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ask}}]) - 1.
\end{aligned}$$

Since $\Pr[\mathsf{GG} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Ask}}] \leq 1/2$, we have

$$\mathsf{Adv}_P^{\mathcal{A}}(k) \leq 2 \cdot \Pr[\mathsf{Forge}] + 2 \cdot \Pr[\mathsf{GG} \wedge \overline{\mathsf{Forge}} \wedge \mathsf{Ask}].$$

Finally, it follows from Equations (2) and (5) that

$$\mathsf{Adv}_P^{\mathcal{A}}(k) \leq 2p_u(k) \cdot \mathsf{Succ}_{\Gamma}(t'') + 2 \cdot \mathsf{Succ}_N(t').$$

This completes the proof of Theorem 1. □

7 Conclusion

In this paper we have presented a dynamic group key agreement scheme. The scheme is simple and practical while meeting strong notions of security. Compared with other provably-secure schemes published up to date, our scheme incurs much lower communication overhead for initial group formation and for group updates, both in terms of the number of communication rounds and the number of messages sent by all users. Due to its communication efficiency, our family of protocols for dynamic group key agreement is well suited for a lossy and high-delay network environment.

References

[1] D.A. Agarwal, O. Chevassut, M.R. Thompson, and G. Tsudik: An Integrated Solution for Secure Group Communication in Wide-Area Networks. In Proc. of 6th IEEE Symposium on Computers and Communications, pp.22–28, 2001.

[2] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik: On the Performance of Group Key Agreement Protocols. In Proc. of 22nd IEEE International Conference on Distributed Computing Systems, pp.463–464, 2002. Full version available at http://www.cnds.jhu.edu/publications/.

[3] G. Ateniese, M. Steiner, and G. Tsudik: New multiparty authentication services and key agreement protocols. IEEE Journal on Selected Areas in Communications, vol.18, no.4, pp.628–639, April 2000.

[4] K. Becker, and U. Wille: Communication complexity of group key distribution. In Proc. of 5th ACM Conf. on Computer and Communications Security, pp.1–6, 1998.

[5] M. Bellare, D. Pointcheval, and P. Rogaway: Authenticated key exchange secure against dictionary attacks, Eurocrypt'00, LNCS 1807, pp.139–155, 2000.

[6] M. Bellare and P. Rogaway: Entity authentication and key distribution. Advances in Cryptology, Crypto'93, LNCS 773, pp.232-249, 1993.

[7] M. Bellare and P. Rogaway: Random oracles are practical: A paradigm for designing efficient protocols. In Proc. of 1st ACM Conf. on Computer and Communications Security (CCS'93), pp.62–73, 1993.

[8] M. Bellare and P. Rogaway: Provably secure session key distribution — the three party case. In Proc. of 27th ACM Symposium on the Theory of Computing (STOC), pp.57–66, 1995.

[9] E. Biham, D. Boneh, and O. Reingold: Breaking generalized Diffie-Hellman modulo a composite is no easier than factoring. Information Processing Letters (IPL), vol.70, no.2, pp.83–87, 1999.

[10] C. Boyd and J.M.G. Nieto: Round-optimal contributory conference key agreement. PKC2003, LNCS 2567, pp.161–174, 2003.

[11] E. Bresson and D. Catalano: Constant round authenticated group key agreement via distributed computation. Proc. 7th International Workshop on Practice and Theory in Public Key Cryptography (PKC'04), LNCS 2947, pp.115–129, 2004.

[12] E. Bresson, O. Chevassut, and D. Pointcheval: Provably authenticated group Diffie-Hellman key exchange — the dynamic case. Asiacrypt'01, LNCS 2248, pp.290–309, 2001.

[13] E. Bresson, O. Chevassut, and D. Pointcheval: Dynamic group Diffie-Hellman key exchange under standard assumptions. Eurocrypt'02, LNCS 2332, pp.321–336, 2002.

[14] E. Bresson, O. Chevassut, and D. Pointcheval: Group Diffie-Hellman key exchange secure against dictionary attacks. Asiacrypt'02, LNCS 2501, pp.497–514, 2002.

[15] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater: Provably authenticated group Diffie-Hellman key exchange. In Proc. of 8th ACM Conf. on Computer and Communications Security, pp.255–264, 2001.

[16] M. Burmester and Y. Desmedt: A secure and efficient conference key distribution system. Eurocrypt'94, LNCS 950, pp.275–286, 1994.

[17] W. Diffie and M.E. Hellman: New Directions in cryptography. IEEE Transactions on Information Theory, vol.22, pp.644-654, 1976.

[18] W. Diffie, P. van Oorschot, and M. Wiener: Authentication and authenticated key exchanges. Designs, Codes, and Cryptography, vol.2, pp.107–125, 1992.

[19] T. ElGamal: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. on Information Theory, vol.31, no.4, pp.469–472, July 1985.

[20] S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal of Computing, vol.17, no.2, pp.281–308, 1988.

[21] I. Ingemarsson, D. Tang, and C. Wong: A conference key distribution system. IEEE Trans. on Information Theory, vol.28, no.5, pp.714–720, September 1982.

[22] M. Just and S. Vaudenay: Authenticated multi-party key agreement. Asiacrypt'96, LNCS 1163, pp.36-49, 1996.

[23] Y. Kim, A. Perrig, and G. Tsudik: Simple and fault-tolerant key agreement for dynamic collaborative groups. In Proc. of 7th ACM Conf. on Computer and Communications Security, pp.235–244, 2000.

[24] Y. Kim, A. Perrig, and G. Tsudik: Communication-efficient group key agreement. In Proc. of International Federation for Information Processing — 16th International Conference on Information Security (IFIP SEC'01), pp.229–244, June 2001.

[25] J. Katz and M. Yung: Scalable protocols for authenticated group key exchange. Crypto'03, LNCS 2729, pp.110–125, August 2003.

[26] O. Pereira and J.-J. Quisquater: A security analysis of the Cliques protocols suites. Proc. 14th IEEE Computer Security Foundations Workshop, pp.73–81, June 2001.

[27] D.G. Steer, L. Strawczynski, W. Diffie, and M. Wiener: A secure audio teleconference system. Crypto'88, LNCS 403, pp.520–528, 1988.

[28] A. Shamir: How to share a secret. Communications of the ACM, vol.22, no.11, pp.612–613, November 1979.

[29] M. Steiner, G. Tsudik, and M. Waidner: Key agreement in dynamic peer groups. IEEE Trans. on Parallel and Distrib. Syst., vol.11, no.8, pp.769–780, August 2000.

[30] W.-G. Tzeng and Z.-J. Tzeng: Round-efficient conference key agreement protocols with provable security. Asiacrypt'00, LNCS 1976, pp.614–627, 2000.