DDH-based Group Key Agreement in a Mobile Environment*

Junghyun Nam, Jinwoo Lee, Seungjoo Kim, and Dongho Won

School of Information and Communication Engineering,
Sungkyunkwan University, 300 Chunchun-dong, Jangan-gu, Suwon,
Gyeonggi-do 440-746, Korea
jhnam@dosan.skku.ac.kr, jwlee@dosan.skku.ac.kr, skim@ece.skku.ac.kr,
dhwon@simsan.skku.ac.kr

Abstract. A group key agreement protocol is designed to efficiently implement secure multicast channels for a group of parties communicating over an untrusted, open network by allowing them to agree on a common secret key. In the past decade many problems related to group key agreement have been tackled and solved (diminished if not solved), and recently some constant-round protocols have been proven secure in concrete, realistic setting. However, all forward-secure protocols so far are still too expensive for small mobile devices. In this paper we propose a new constant-round protocol well suited for a mobile environment and prove its security under the Decisional Diffie-Hellman assumption. The protocol meets simplicity, efficiency, and all the desired security properties.

Keywords: group key agreement, multicast, security model, DDH

1 Introduction

Over the past several years there has been a tremendous surge of interest in mobile computing. Advances in the power of mobile devices, such as personal digital assistants (PDAs), smart phones and handheld computers, have opened tempting new opportunities for a broad range of communities and institutions. With prices reducing and functionality increasing, it is expected that these network-enabled devices will play major roles in the promotion of both personal and business productivity. It is clear, thus, that the next generation of communication networks will include rapid deployments of independent mobile users.

Although mobile devices are of increasing importance in every aspect of our daily lives, security is still a major gating factor for their full adoption. Despite all the work conducted over many decades, the implementation of strong protection in a mobile environment is non-trivial [14]. Security solutions targeted for more traditional networks are often not directly applicable to wireless networks due to a marked difference in computing resources between mobile and stationary hosts.

One typical example are protocols for group key agreement, which are designed to efficiently implement secure multicast channels for a group of parties communicating over a public network by providing them with a shared secret key called a *session key*. Although some constant-round protocols for group key agreement have been proposed [13, 23, 7], they are still too costly to be practical for applications involving mobile devices with limited computing resources; in these protocols, the computational cost of each group member increases significantly as group size grows. Other constant-round protocols [6, 11], while they require only a fixed amount of computation for all but one group member, do not provide

* An updated version of this paper will appear in the Journal of Systems and Software.

perfect forward secrecy [16]. It is this observation that prompted the present work aimed at designing an efficient group key agreement protocol not only meeting strong notions of security, but also well suited for a mobile environment.

The mobile computing architecture we visualize is asymmetric in the sense of computational capabilities of hosts. That is, the protocol participants consist of a stationary host and a cluster of mobile hosts. The stationary host (also called *server*) with sufficient computational power and mobile hosts (also called *clients*) with limited computational resources wish to communicate securely with each other by agreeing on a common session key [11, 27].

1.1 Our Contribution

Our group key agreement protocol is provably secure against a powerful active adversary who controls all communication flows in the network and even executes an unbounded number of concurrent instances of the protocol. We provide a rigorous proof of security under the well-known Decisional Diffie-Hellman (DDH) assumption in a formal security model which improves that of Bresson et al. [12]. Furthermore, in contrast with other asymmetric protocols [6, 11] with provable security, our group key agreement protocol provides perfect forward secrecy; i.e., disclosure of long-term secret keys does not compromise the security of previously established session keys.

Despite meeting all these strong notions of security, our construction is surprisingly simple and provides a practical solution for group key agreement in a mobile environment similar to our setting. In a protocol execution involving mobile hosts, a bottleneck arises when the number of public-key cryptography operations that need to be performed by a mobile host increases accordingly as group size grows. It is therefore of prime importance for a group key agreement protocol to offer a low, fixed amount of computations to its mobile participants. To this end our protocol shifts much of the computational burden to the server with sufficient computational power. By allowing this computational asymmetry among protocol participants (as also can be observed in the previous works [6, 11]), the computational cost of a mobile participant of our protocol is reduced to two modular exponentiations (plus one signature generation and verification) without respect to the number of participants.

In addition our group key agreement protocol is very efficient in terms of the number of communication rounds; it requires only three rounds of communication among participants. Keeping the number of communication rounds constant is critical for efficient and scalable group key agreement particularly over a wide area network, where the dominant source of delay is the communication time spent in the network rather than the computational time needed for cryptographic operations [1, 22].

As an additional contribution, we propose a refinement of the standard security model of Bresson et al. [12], which we believe to be an issue of independent interest. As shown in Section 5, our refinement greatly simplifies the security proof of the compiler presented by Katz and Yung [23] even in the presence of a stronger adversary.

1.2 Related Work

The original idea of extending the 2-party Diffie-Hellman scheme [15] to the multi-party setting dates back to the classical paper of Ingemarsson et al. [19], and is followed by many works [25, 13, 20, 3, 21, 26, 22] offering various levels of complexity. However, research on provably-secure group key agreement in concrete, realistic setting is fairly new. It is only recently that Bresson et al. [12, 8, 9] have presented the first group key agreement protocols proven secure in a well-defined security model which builds on earlier model of Bellare et al. [4]. The initial work [12] assumes that group membership is static, whereas later works [8, 9] focus on the dynamic case which we do not deal with here. But one drawback of their scheme is that its round complexity is linear in the number of group members. Consequently, as group size grows large, this scheme becomes impractical particularly in a wide area network with high communication latency.

More recently, Katz and Yung [23] have proposed the first constant-round group key agreement protocol that has been proven secure in the security model of Bresson et al. [12]. They provide a formal proof of security for the two-round protocol of Burmester and Desmedt [13], and introduce a one-round compiler that transforms any group key agreement protocol secure against a passive adversary into one that is secure against an active adversary. 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, $O(n)$ signature verifications, and two signature generations. While this protocol is very efficient in general, the full symmetry negatively impacts on the overall performance of the protocol in our asymmetric setting; the computational cost of a mobile host is significant in a large group, due to the number of modular multiplications and signature verifications.

Most recently, Bresson and Catalano [7] have introduced another fully-symmetric protocol which requires two rounds of communication. Interestingly, unlike previous approaches, they construct the protocol by combining the properties of the ElGamal encryption scheme [17] with standard secret sharing techniques [24]. However, with increasing number of participants, the complexity of the protocol becomes beyond the capabilities of a small mobile device.

The protocol presented by Boyd and Nieto [6] completes in only a single round of communication and is provably secure in the random oracle model [5]. 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 one-round group key agreement protocol providing forward secrecy. Another constant-round protocol that does not achieve (perfect) forward secrecy has been shown in [11]. This protocol runs in two rounds of communication and is provably secure in the random oracle model. In common with our protocol, these protocols [6, 11] are computationally asymmetric; one distinct member performs $O(n)$ computations whereas the other members perform only a constant amount of computation.

1.3 Outline

The remainder of this paper is organized as follows. In Section 2, we begin with a description of our security model for group key agreement protocols. In Section 3, we first define the security of a group key agreement protocol and then describe the underlying assumptions on which the security of our protocol rests. Finally, we present a two-round group key agreement protocol secure against a passive adversary and a three-round group key agreement protocol secure against an active adversary in Section 4 and Section 5, respectively.

2 The Model

In this section we refine the formal security model which has been widely used in the literature [12, 8–10, 23, 6] to analyze group key agreement protocols. In particular, we incorporate *strong corruption* [4] into the security model in a different way than the previous approaches by allowing an adversary to ask one additional query, Dump, and we modify the definition of *freshness* according to the refined model. Section 5 shows that our approach leads to much simpler security proof of the compiler presented by Katz and Yung [23].

Participants. Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be a set of n users who wish to participate in a group key agreement protocol P. The number of users, n, is polynomially bounded in the security parameter k. Users may execute the protocol multiple times concurrently and thus each user can have many instances called oracles. We use Π_i^s to denote instance s of user U_i. 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 $\mathcal{A}$.

Partners. Informally, the *partners* of oracle Π_i^s (denoted PID_i^s) is the set of all the instances that should compute the same session key as Π_i^s in an execution of the protocol. Before defining PID_i^s formally, we first define the session ID for oracle Π_i^s which we denote by SID_i^s. In an execution of the protocol, let $\mathcal{P}_i^s$ is the set of all oracles with which oracle Π_i^s has exchanged some messages, and M_{ij}^{st} is the concatenation of all messages that oracle Π_i^s has exchanged with oracle Π_j^t. Then we define SID_i^s as

$$\mathsf{SID}_i^s = \{M_{ij}^{st} \mid \Pi_j^t \in \mathcal{P}_i^s\}.$$

Let ACC_i^s be a variable that is $\mathtt{TRUE}$ if Π_i^s has computed a session key, and $\mathtt{FALSE}$ otherwise. Then, using the session ID defined above, PID_i^s is defined as follows:

$$\begin{aligned}\mathsf{PID}_i^s = \{\Pi_j^t \mid\ & \mathsf{SID}_i^s \cap \mathsf{SID}_k^u \neq \emptyset \wedge \mathsf{SID}_k^u \cap \mathsf{SID}_j^t \neq \emptyset \wedge \\ & \mathsf{ACC}_i^s = \mathsf{ACC}_k^u = \mathsf{ACC}_j^t = \mathtt{TRUE},\ \text{for some } \Pi_k^u\}.\end{aligned}$$

Note that in the above definition of PID_i^s, it is possible that $\Pi_i^s = \Pi_k^u$. Therefore, the conjunction simply says that oracle Π_j^t is a partner of oracle Π_i^s if $\mathsf{SID}_i^s \cap \mathsf{SID}_j^t \neq \emptyset$ and $\mathsf{ACC}_i^s = \mathsf{ACC}_j^t = \mathtt{TRUE}$, or they share the same partner. 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 the following various queries, each of which captures a capability of the adversary.

- $\mathsf{Execute}(\mathcal{U})$: This query returns a transcript of an honest protocol execution among instances of the users in $\mathcal{U}$.
- $\mathsf{Send}(\Pi_i^s, m)$: This query sends message m to oracle Π_i^s. When oracle Π_i^s receives the message m, it proceeds as specified in the protocol; the oracle updates its state, and generates and sends out a response message as needed. The response message, if any, is returned to the adversary $\mathcal{A}$. A query of the form $\mathsf{Send}(\Pi_i^s, \text{“}\mathtt{start}\text{”})$ allows adversary $\mathcal{A}$ to initiate an execution of the protocol.

- $\mathsf{Reveal}(\Pi_i^s)$: This query returns the session key K of oracle Π_i^s.
- $\mathsf{Corrupt}(U_i)$: This query returns the long-term private key SK_i of user U_i.
- $\mathsf{Dump}(\Pi_i^s)$: This query returns *all short-term secret values* of oracle Π_i^s.
- $\mathsf{Test}(\Pi_i^s)$: This query is asked only once when the adversary $\mathcal{A}$ wants to attempt to distinguish the real session key K from a random fake key. To answer the query, 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 only if oracle Π_i^s is *fresh*, the definition of which will be given below.

Definition 1. *Oracle Π_i^s is said to be* fresh *if all of the following conditions hold:*

1. $\mathsf{ACC}_i^s = \mathtt{TRUE}$.
2. *No one in PID_i^s has been asked for a* Reveal *query (note that $\Pi_i^s \in \mathsf{PID}_i^s$ unless $\mathsf{PID}_i^s \neq \emptyset$).*
3. *No one in $\mathcal{U}$ has been asked for a* Corrupt *query before the number of partners of Π_i^s, $|\mathsf{PID}_i^s|$, becomes equal to n.*
4. *No one in $\mathcal{P}_i^s$ has been asked for a* Dump *query.*

Definition 2. *An adversary is called* active *if it makes all the queries above, and is called* passive *if it makes only five of them:* Execute, Reveal, Corrupt, Dump, *and* Test.

3 Security Definitions

In this section we first define the security of a group key agreement protocol and then describe the cryptographic assumptions on which the security of our protocol is based.

Group Key Agreement. The security of a group key agreement protocol P is defined in the following context. The adversary $\mathcal{A}$ executes the protocol exploiting as much parallelism as possible and any queries allowed in the security model. During executions of the protocol, the adversary $\mathcal{A}$, at any time, asks a Test query to a fresh oracle, gets back an ℓ-bit string as the response to this query, and at some later point, outputs a bit b' as a guess for the hidden bit b. Let CG ($\mathsf{Correct\ Guess}$) be the event that $b' = b$. Then we define the advantage of $\mathcal{A}$ in attacking protocol P to be

$$\mathsf{Adv}_{\mathcal{A},P}(k) = 2 \cdot \Pr[\mathsf{CG}] - 1.$$

We say that protocol P is secure against an adversary $\mathcal{A}$ if $\mathsf{Adv}_{\mathcal{A},P}(k)$ is negligible. Furthermore, we say that protocol P is a secure group key agreement protocol if it is secure against all probabilistic polynomial time adversaries $\mathcal{A}$.

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 $\mathsf{Succ}_{\mathcal{A},\Gamma}(k)$ the probability of an adversary $\mathcal{A}$ succeeding with an existential forgery under adaptive chosen message attack [18]. We say that a signature scheme Γ is secure if $\mathsf{Succ}_{\mathcal{A},\Gamma}(k)$ is negligible for any probabilistic polynomial time adversary $\mathcal{A}$. We denote by $\mathsf{Succ}_{\Gamma}(t)$ the maximum value of $\mathsf{Succ}_{\mathcal{A},\Gamma}(k)$ over all adversaries $\mathcal{A}$ running in time at most t.

DDH Assumption. Let $\mathbb{G} = \langle g \rangle$ be any finite cyclic group of prime order q and let x, y, z be randomly chosen elements in $\mathbb{Z}_q$. Informally, the DDH assumption is that it is difficult to distinguish between the distributions of (g^x, g^y, g^{xy}) and (g^x, g^y, g^z). More formally, if we define $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A})$ as

$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A}) = \Big|\Pr[\mathcal{A}(g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(g, g^x, g^y, g^z) = 1]\Big|,$$

we say that the DDH assumption holds in $\mathbb{G}$ if $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A})$ is negligible for any probabilistic polynomial time adversary $\mathcal{A}$. We denote by $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$ the maximum value of $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A})$ over all adversaries $\mathcal{A}$ running in time at most t.

4 A Two-Round Group Key Agreement Protocol

We now present a group key agreement protocol P_1 secure against a passive adversary under the DDH assumption. The public parameters $\mathbb{G}$ and g, as defined in Section 3, are assumed to be known in advance to all parties. Then the protocol P_1 runs in two rounds, one with $n-1$ unicasts and the other with a single broadcast, as follows:

1. Each user (mobile host or client) $U_i \neq U_n$ chooses a random $r_i \in \mathbb{Z}_q$, computes $z_i = g^{r_i}$, and sends $m_i = (U_i, z_i)$ to the stationary host (or the server) U_n, who chooses random $r, r_n \in \mathbb{Z}_q$ and computes $z = g^r$ and $z_n = g^{r_n}$.
2. Having computed $X = \prod_{i \in [1,n]} x_i$ and the set $Y = \{y_i \mid 1 \leq i \leq n-1\}$, where $x_i = z_i^r$ and $y_i = X \cdot x_i^{-1}$, the server U_n broadcasts $m_n = (U_n, z, Y)$ to the entire group.
3. Upon receiving the broadcast, each $U_i \neq U_n$ computes $X = y_i \cdot z^{r_i}$. All users in $\mathcal{U}$ compute their session key as $K = H(Y, X)$, where H is a one-way hash function modelled as a random oracle in the security proof.

Suppose, for example, that $\mathcal{U} = \{U_1, U_2, U_3, U_4\}$. Then the server U_4 receives $\{g^{r_1}, g^{r_2}, g^{r_3}\}$ from clients, and broadcasts g^r and $Y = \{g^{r(r_2+r_3+r_4)}, g^{r(r_1+r_3+r_4)}, g^{r(r_1+r_2+r_4)}\}$. All users in $\mathcal{U}$ compute the same key: $K = H(Y, X)$, where $X = g^{r(r_1+r_2+r_3+r_4)}$.

Note that in the protocol above, the server does not need to wait for the last message from clients before it can start to perform the computation. Furthermore, if precomputations are possible, all the exponentiations in the first round can be performed off-line and thus, only 1 exponentiation per client is required to be done on-line.

Theorem 1. *Let $\mathcal{A}$ be a passive adversary attacking protocol P_1, running in time t and making q_{ex}* $\mathsf{Execute}$ *queries. Then we have*

$$\mathsf{Adv}_{\mathcal{A},P_1}(k) \leq 2q_{ex} \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t'),$$

where $t' = t + O(nq_{ex}t_{exp})$ and t_{exp} is the time required to compute an exponentiation in $\mathbb{G}$.

Proof. Assume that $\mathcal{A}$ can guess the hidden bit b correctly with probability $1/2 + \epsilon$. Then we construct from $\mathcal{A}$ a distinguisher $\mathcal{D}$ that solves the DDH problem in $\mathbb{G}$ with probability ϵ/q_{ex}.

Before describing the construction of $\mathcal{D}$, let us first define the following two distributions:

$$\mathsf{Real} = \left\{ (T,K) \;\middle|\; \begin{array}{l} r_1,\dots,r_n,r \in_R \mathbb{Z}_q; \\ z_1 = g^{r_1},\dots,z_n = g^{r_n}, z = g^r; \\ x_1 = g^{rr_1},\dots,x_n = g^{rr_n}; \\ X = x_1 \cdots x_n; \\ y_1 = X \cdot x_1^{-1},\dots,y_{n-1} = X \cdot x_{n-1}^{-1} \end{array} \right\},$$

$$\mathsf{Rand} = \left\{ (T,K) \;\middle|\; \begin{array}{l} r_1,\dots,r_n,r,s_1,\dots,s_n \in_R \mathbb{Z}_q; \\ z_1 = g^{r_1},\dots,z_n = g^{r_n}, z = g^r; \\ x_1 = g^{s_1},\dots,x_n = g^{s_n}; \\ X = x_1 \cdots x_n; \\ y_1 = X \cdot x_1^{-1},\dots,y_{n-1} = X \cdot x_{n-1}^{-1} \end{array} \right\},$$

where $T = (z, z_1, \dots, z_{n-1}, y_1, \dots, y_{n-1})$ and $K = H(y_1, \dots, y_{n-1}, X)$.

Lemma 1. *Let $\mathcal{A}'$ be an algorithm that, given (T,K) coming from one of the two distributions* Real *and* Rand*, runs in time t and outputs 0 or 1. Then we have:*

$$\begin{aligned} \big|\Pr[\mathcal{A}'(T,K) = 1 \mid (T,K) \leftarrow \mathsf{Real}] - \\ \Pr[\mathcal{A}'(T,K) = 1 \mid (T,K) \leftarrow \mathsf{Rand}]\big| \\ \le \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t + (4n-6)t_{exp}). \end{aligned}$$

Proof. We prove the lemma by using the random self-reducibility of the DDH problem. Consider the following distribution, which is constructed from the triple $(g^r, g^{r_2}, g^{r'r_2}) \in \mathbb{G}^3$:

$$\mathsf{Dist} = \left\{ (T,K) \;\middle|\; \begin{array}{l} r_1,\alpha_3,\beta_3,\dots,\alpha_n,\beta_n \in_R \mathbb{Z}_q; \\ z_1 = g^{r_1}, z_2 = g^{r_2}, \\ z_3 = g^{r_1\alpha_3 + r_2\beta_3},\dots,z_n = g^{r_1\alpha_n + r_2\beta_n}, z = g^r; \\ x_1 = g^{rr_1}, x_2 = g^{r'r_2}, \\ x_3 = g^{rr_1\alpha_3 + r'r_2\beta_3},\dots,x_n = g^{rr_1\alpha_n + r'r_2\beta_n}; \\ X = x_1 \cdots x_n; \\ y_1 = X \cdot x_1^{-1},\dots,y_{n-1} = X \cdot x_{n-1}^{-1} \end{array} \right\},$$

where T and K are as defined above. If $(g^r, g^{r_2}, g^{r'r_2})$ is a Diffie-Hellman triple (i.e., $r = r'$), we have $\mathsf{Dist} \equiv \mathsf{Real}$ since $x_i = z_i^r$ for all $i \in [1,n]$. If instead $(g^r, g^{r_2}, g^{r'r_2})$ is a random triple, it is clear that $\mathsf{Dist} \equiv \mathsf{Rand}$. □

Lemma 2. *For any (computationally unbounded) adversary $\mathcal{A}$, we have:*

$$\Pr[\mathcal{A}(T, K_b) = b \mid (T, K_1) \leftarrow \mathsf{Rand}; K_0 \leftarrow \{0,1\}^{\ell}; b \leftarrow \{0,1\}] = 1/2.$$

Proof. In experiment Rand, the transcript T constrains the values s_i by the following $n-1$ equations:

$$\begin{aligned} \log_g y_1 &= -s_1 + \sum_{i=1}^{n} s_i, \\ \log_g y_2 &= -s_2 + \sum_{i=1}^{n} s_i, \\ &\vdots \\ \log_g y_{n-1} &= -s_{n-1} + \sum_{i=1}^{n} s_i. \end{aligned}$$

Since T does not constrain the values s_i any further and since the equation $\log_g X = \sum_{i=1}^{n} s_i$ is not expressible as a linear combination of the $n-1$ equations above, we have that the value of X is independent of T. This implies that

$$\Pr[\mathcal{A}(T, X_b) = b \mid (T, X_1) \leftarrow \mathsf{Rand}; X_0 \leftarrow \mathbb{G}; b \leftarrow \{0,1\}] = 1/2.$$

Then, since H is a random oracle, the statement of Lemma 2 immediately follows. □

Armed with the two lemmas above, we now give the details of the construction of the distinguisher $\mathcal{D}$. Assume without loss of generality that $\mathcal{A}$ makes its Test query to an oracle activated by the δ^{th} $\mathsf{Execute}$ query. The distinguisher $\mathcal{D}$ begins by choosing a random $d \in \{1, \ldots, q_{ex}\}$ as a guess for the value of δ. $\mathcal{D}$ then invokes $\mathcal{A}$ and simulates the queries of $\mathcal{A}$. $\mathcal{D}$ answers all the queries from $\mathcal{A}$ in the obvious way, following the protocol exactly as specified, except if a query is the d^{th} $\mathsf{Execute}$ query. In this latter case, $\mathcal{D}$ slightly deviates from the protocol, by embedding the DDH problem instance given as input into the transcript as follows.

Given a triple $(g^r, g^{r_2}, g^{r' r_2}) \in \mathbb{G}^3$, $\mathcal{D}$ generates (T, K) according to the the distribution Dist and answers the d^{th} $\mathsf{Execute}$ query of $\mathcal{A}$ with T. The distinguisher $\mathcal{D}$ aborts and outputs a random bit if $d \neq \delta$. Otherwise, $\mathcal{D}$ answers the Test query of $\mathcal{A}$ with K. At some later point, when $\mathcal{A}$ terminates and outputs its guess b', $\mathcal{D}$ outputs 1 if $b = b'$, and 0 otherwise. By Lemma 1 and 2, and since $\Pr[d = \delta] = 1/q_{ex}$ and

$$\Pr[\mathcal{A}(T, K_b) = b \mid (T, K_1) \leftarrow \mathsf{Real}; K_0 \leftarrow \{0,1\}^{\ell}; b \leftarrow \{0,1\}] = 1/2 + \epsilon,$$

we obtain

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{D}) = \epsilon / q_{ex},$$

which immediately yields the statement of Theorem 1. □

5 A Three-Round Group Key Agreement Protocol

In this section we propose a group key agreement protocol P_2 secure against an active adversary. We transform protocol P_1 to the protocol P_2 by applying a variant of the compiler presented in [23]. The protocol P_2 proceeds as follows:

1. Each user U_i chooses an instance identifier (IID) $\phi_i \in_R \{0,1\}^k$ and broadcasts $\{U_i, \phi_i\}$. Having received all the $n-1$ IIDs from other users, each U_i sets $\Phi_i = \{\{U_i, \phi_i\} \mid 1 \leq i \leq n\}$.
2. The users in $\mathcal{U}$ now proceed as specified in the protocol P_1, except that: (1) each user U_i sends $m'_i = (m_i, \sigma_i)$ instead of m_i, where σ_i is the signature of $m_i \| \Phi_i$, and (2) upon receiving message $m'_j = (m_j, \sigma_j)$ from user U_j, U_i verifies that $\mathcal{V}_{pk_j}(m_j \| \Phi_i, \sigma_j) = 1$. All users in $\mathcal{U}$ compute their session key as in P_1.

Theorem 2. *Let $\mathcal{A}_2$ be an active adversary attacking protocol P_2, running in time t and making q_{ex}* Execute *queries and q_{se}* Send *queries. Let $\mathsf{Adv}_{P_1}(t', q_{ex} + \frac{q_{se}}{n})$ be the maximum advantage in attacking protocol P_1, where the maximum is over all passive adversaries that run in time t' and make $q_{ex} + \frac{q_{se}}{n}$* Execute *queries. Then we have*

$$\mathsf{Adv}_{\mathcal{A}_2,P_2}(k) \leq \mathsf{Adv}_{P_1}(t', q_{ex} + \frac{q_{se}}{n}) + n \cdot \mathsf{Succ}_{\Gamma}(t'') + \frac{q_{se}^2 + q_{ex}q_{se}}{2^k},$$

where $t' = t + O(nq_{ex}t_{exp} + nq_{se}t_{exp})$, $t'' = t + O(nq_{ex}t_{exp} + q_{se}t_{exp})$, and t_{exp} is as in Theorem 1.

Proof. The proof of the theorem proceeds by constructing from $\mathcal{A}_2$ a passive adversary $\mathcal{A}_1$ attacking protocol P_1. Before describing the details of the construction, we first bound the probability of the event, Forge, that $\mathcal{A}_2$ outputs a valid forgery with respect to the public key PK_i of some user $U_i \in \mathcal{U}$ before making the query $\mathsf{Corrupt}(U_i)$.

Lemma 3. $\Pr[\mathsf{Forge}] \leq n \cdot \mathsf{Succ}_{\Gamma}(t'')$*, where t'' is as in Theorem 2.*

Proof. We build from $\mathcal{A}_2$ a signature forger $\mathcal{F}$ against the signature scheme Γ. The goal of the forger $\mathcal{F}$, given as input a public key PK and access to a signing oracle associated with this key, is to output a valid forgery (m, σ) with respect to PK, i.e., $\mathcal{V}_{PK}(m, \sigma) = 1$ such that σ was not previously output by the signing oracle as a signature on the message m. The forger $\mathcal{F}$ 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 have $\mathcal{A}_2$ run, simulating the queries from $\mathcal{A}_2$ as follows:

- $\mathsf{Execute}(\mathcal{U})/\mathsf{Reveal}(\Pi_i^s)/\mathsf{Dump}(\Pi_i^s)/\mathsf{Test}(\Pi_i^s)$: These queries are answered in the obvious way.
- $\mathsf{Send}(\Pi_i^s, m)$: If $i \neq f$, $\mathcal{F}$ knows the private signing key of U_i, and hence can answer the queries following the protocol 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.
- $\mathsf{Corrupt}(U_i)$: If $i \neq f$, $\mathcal{F}$ simply hands the private key SK_i which was generated by $\mathcal{F}$ itself. If instead $\mathcal{A}_2$ corrupts $U_i = U_f$, then $\mathcal{F}$ halts and outputs "`fail`".

The simulation provided above is perfectly indistinguishable from the real execution unless adversary $\mathcal{A}_2$ makes the query $\mathsf{Corrupt}(U_f)$. Throughout this simulation, $\mathcal{F}$ monitors each Send query from $\mathcal{A}_2$, and checks if it includes a valid message/signature pair (m, σ) with respect to PK. If no such query is made until $\mathcal{A}_2$ stops, then $\mathcal{F}$ halts and outputs "`fail`". Otherwise, $\mathcal{F}$ outputs (m, σ) as a valid forgery with respect to PK. Lemma 3 directly follows from the fact that this latter case occurs with probability $\Pr[\mathsf{Forge}]/n$. □

We now describe the construction of the passive adversary $\mathcal{A}_1$ in detail. After generating a public/private key pair (PK_i, SK_i) for each $U_i \in \mathcal{U}$, the adversary $\mathcal{A}_1$ invokes $\mathcal{A}_2$ and simulates the queries of $\mathcal{A}_2$ as follows.

$\mathsf{Execute}(\mathcal{U})$: $\mathcal{A}_1$ issues its own $\mathsf{Execute}$ query to get a transcript T_1 of an execution of P_1. $\mathcal{A}_1$ then generates a transcript T_2 of an execution of P_2, by choosing random $\phi_1, \ldots, \phi_n \in \{0,1\}^k$, signing the messages in T_1, and prepending $\Phi = \{\{U_i, \phi_i\} \mid 1 \le i \le n\}$ to this signed transcript. Finally, $\mathcal{A}_1$ returns T_2 as the answer to the $\mathsf{Execute}$ query of $\mathcal{A}_2$ and adds (Φ, T_1) into a list $\mathcal{L}$ which is maintained by $\mathcal{A}_1$ to link a simulated execution of P_2 to an execution of P_1.

$\mathsf{Send}(\Pi_i^s, m)$: If some user in $\mathcal{U}$ has been asked for a $\mathsf{Corrupt}$ query before this query, then $\mathcal{A}_1$ handles the query in the obvious way following the protocol P_2 exactly as specified. Otherwise, $\mathcal{A}_1$ simulates the query as follows, using the similar way as it did for $\mathsf{Execute}$ queries:

If $m =$ "`start`", $\mathcal{A}_1$ chooses a random $\phi_i^s \in \{0,1\}^k$ and returns $\{U_i, \phi_i^s\}$ to $\mathcal{A}_2$. After receiving all the expected IIDs in the first round, $\mathcal{A}_1$ defines Φ_i^s as per protocol specification. If $\mathcal{A}_1$ needs to return the message m_i' in response to this Send query, $\mathcal{A}_1$ first checks the list $\mathcal{L}$ to see if there exists an entry of the form (Φ_i^s, T_1). If so, then $\mathcal{A}_1$ generates the message m_i' from the message m_i in T_1 and returns it to adversary $\mathcal{A}_2$. Otherwise, $\mathcal{A}_1$ obtains a transcript T_1 of an execution of P_1 by making an $\mathsf{Execute}$ query, adds the pair (Φ_i^s, T_1) to the list $\mathcal{L}$, and then proceeds as in the former case.

$\mathsf{Dump}(\Pi_i^s)$: Let T_1 be the transcript such that $(\Phi_i^s, T_1) \in \mathcal{L}$. Then, $\mathcal{A}_1$ makes a Dump query to the U_i's instance activated by the $\mathsf{Execute}$ query that resulted in the transcript T_1, and simply forwards the random secret exponent(s) obtained from this Dump query.

$\mathsf{Reveal}(\Pi_i^s)$: As can be seen from the way $\mathcal{A}_1$ handles $\mathsf{Execute}$ and Send queries of $\mathcal{A}_2$, the session key of Π_i^s is unavailable to $\mathcal{A}_1$ unless some Dump queries or $\mathsf{Corrupt}$ queries have been asked by $\mathcal{A}_2$. However, this Reveal query can be answered as follows:

1. Suppose that no one in $\mathcal{U}$ has been asked for a $\mathsf{Corrupt}$ query before Π_i^s receives its last incoming message. Let T_1 be the transcript such that $(\Phi_i^s, T_1) \in \mathcal{L}$. Then, $\mathcal{A}_1$ asks a Reveal query to the U_i's instance activated by the $\mathsf{Execute}$ query that resulted in the transcript T_1, and forwards the result of this Reveal query to adversary $\mathcal{A}_2$.
2. Now suppose that some user in $\mathcal{U}$ has been asked for a $\mathsf{Corrupt}$ query before Π_i^s receives its last incoming message. Note that in this case, $\mathcal{A}_2$ may have signed and sent arbitrary messages of its choice to Π_i^s. We further separate this case into the following two subcases:
   - Consider the case that $i \neq n$ and $\mathcal{A}_2$ has made a $\mathsf{Corrupt}$ query to U_n after Π_i^s has sent the message m_i' and before Π_i^s has received the message m_n'. In this case $\mathcal{A}_1$ first obtains the random secret exponent by making its own Dump query in the same way as it did for Dump queries of $\mathcal{A}_2$. $\mathcal{A}_1$ then computes the session key of Π_i^s using this random exponent and returns the result to adversary $\mathcal{A}_2$.
   - For other cases, $\mathcal{A}_1$ has already the random secret exponent(s) for Π_i^s which were chosen by $\mathcal{A}_1$ itself, and thus can answer the query following the protocol exactly as specified.

$\mathsf{Corrupt}(U_i)$: $\mathcal{A}_1$ simply returns the long-term private key SK_i of U_i.

$\mathsf{Test}(\Pi_i^s)$: $\mathcal{A}_1$ finds a pair $(\Phi_i^s, T_1) \in \mathcal{L}$, asks a Test query to one of the oracles activated by the $\mathsf{Execute}$ query that resulted in T_1, and returns the ℓ-bit string received as the response to its Test query.

Before quantifying the advantage of $\mathcal{A}_1$ in attacking the protocol P_1, we first need to define the event Same. Let Same be the event that a same IID is used by a user to identify two different instances, one activated by a Send query and the other activated by either an $\mathsf{Execute}$ or a Send query. Then, a straightforward calculation shows that

$$\Pr[\mathsf{Same}] \le \frac{q_{se}^2 + q_{se}q_{ex}}{2^k}. \tag{1}$$

During the simulation above, $\mathcal{A}_1$ simply aborts and outputs a random bit if Same or Forge occurs. Otherwise, $\mathcal{A}_1$ outputs whatever $\mathcal{A}_2$ does. Note that as long as neither Same nor Forge occur, the simulation provided by $\mathcal{A}_1$ is perfectly indistinguishable from a real execution of P_2, and in a particular session, $\mathcal{A}_2$ is limited to send messages generated by $\mathcal{A}_1$ from one same transcript of an execution of P_1. This implies that

$$\Pr_{\mathcal{A}_1,P_1}[\mathsf{CG}] = \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Same}}] + \frac{1}{2}\Pr[\mathsf{Forge} \vee \mathsf{Same}]. \tag{2}$$

Using Eq. (2), a simple probability calculation shows that

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A}_2,P_2}(k) &= 2 \cdot \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG}] - 1 \\
&= 2 \cdot \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG} \wedge \mathsf{Forge}] + 2 \cdot \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG} \wedge \overline{\mathsf{Forge}}] - 1 \\
&\le 2 \cdot \Pr[\mathsf{Forge}] + 2 \cdot \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG} \wedge \overline{\mathsf{Forge}}] - 1 \\
&= 2 \cdot \Pr[\mathsf{Forge}] + 2 \cdot \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG} \wedge \overline{\mathsf{Forge}} \wedge \mathsf{Same}] \\
&\quad + 2 \cdot \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Same}}] - 1 \\
&= 2 \cdot \Pr[\mathsf{Forge}] + 2 \cdot \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG} \wedge \overline{\mathsf{Forge}} \wedge \mathsf{Same}] \\
&\quad - \Pr[\mathsf{Forge} \vee \mathsf{Same}] + 2 \cdot \Pr_{\mathcal{A}_1,P_1}[\mathsf{CG}] - 1.
\end{aligned}$$

Since $\Pr[\mathsf{Forge} \vee \mathsf{Same}] \ge \Pr[\mathsf{Forge}] + \Pr[\mathsf{CG} \wedge \overline{\mathsf{Forge}} \wedge \mathsf{Same}]$, it follows that

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A}_2,P_2}(k) &\le \mathsf{Adv}_{\mathcal{A}_1,P_1}(k) + \Pr[\mathsf{Forge}] + \Pr_{\mathcal{A}_2,P_2}[\mathsf{CG} \wedge \overline{\mathsf{Forge}} \wedge \mathsf{Same}] \\
&\le \mathsf{Adv}_{P_1}(t') + \Pr[\mathsf{Forge}] + \Pr[\mathsf{Same}].
\end{aligned}$$

Combined with Lemma 3 and Eq. (1), this immediately yields the desired result. □

6 Conclusion

In this paper we have proposed an efficient, asymmetric group key agreement protocol well suited for groups consisting of a cluster of mobile hosts with limited computational resources and a stationary host with sufficient computational power. The protocol achieves perfect forward secrecy and has been proven secure against an active adversary in the random oracle model under the Decisional Diffie-Hellman assumption.

References

1. Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik: On the Performance of Group Key Agreement Protocols, in: Proceedings of 22nd IEEE International Conference on Distributed Computing Systems, pp. 463–464, 2002. Full version available at http://www.cnds.jhu.edu/publications/.
2. 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.
3. K. Becker, and U. Wille: Communication complexity of group key distribution, in: Proceedings of 1st ACM Conference on Computer and Communications Security (CCS'98), pp. 1–6, 1998.
4. M. Bellare, D. Pointcheval, and P. Rogaway: Authenticated key exchange secure against dictionary attacks, in: Advances in Cryptology – Eurocrypt'00, LNCS 1807, pp. 139–155, 2000.
5. M. Bellare and P. Rogaway: Random oracles are practical: A paradigm for designing efficient protocols, in: Proceedings of 1st ACM Conference on Computer and Communications Security (CCS'93), pp. 62–73, 1993.
6. C. Boyd and J.M.G. Nieto: Round-optimal contributory conference key agreement, in: Proceedings of 6th International Workshop on Practice and Theory in Public Key Cryptography (PKC'03), LNCS 2567, pp. 161–174, 2003.
7. E. Bresson and D. Catalano: Constant round authenticated group key agreement via distributed computation, in: Proceedings of 7th International Workshop on Practice and Theory in Public Key Cryptography (PKC'04), LNCS 2947, pp. 115–129, 2004.
8. E. Bresson, O. Chevassut, and D. Pointcheval: Provably authenticated group Diffie-Hellman key exchange — the dynamic case, in: Advances in Cryptology – Asiacrypt'01, LNCS 2248, pp. 290–309, 2001.
9. E. Bresson, O. Chevassut, and D. Pointcheval: Dynamic group Diffie-Hellman key exchange under standard assumptions, in: Advances in Cryptology – Eurocrypt'02, LNCS 2332, pp. 321–336, 2002.
10. E. Bresson, O. Chevassut, and D. Pointcheval: Group Diffie-Hellman key exchange secure against dictionary attacks, in: Advances in Cryptology – Asiacrypt'02, LNCS 2501, pp. 497–514, 2002.
11. E. Bresson, O. Chevassut, A. Essiari and D. Pointcheval: Mutual authentication and group key agreement for low-power mobile devices, in: Proceedings of the 5th IFIP-TC6 International Conference on Mobile and Wireless Communications Networks (MWCN'03), pp. 59–62, 2003.
12. E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater: Provably authenticated group Diffie-Hellman key exchange, in: Proceedings of 1st ACM Conference on Computer and Communications Security (CCS'01), pp. 255–264, 2001.
13. M. Burmester and Y. Desmedt: A secure and efficient conference key distribution system, in: Advances in Cryptology – Eurocrypt'94, LNCS 950, pp. 275–286, 1994.
14. N. Borisov, I. Goldberg, and D. Wagner: Intercepting mobile communications: The insecurity of 802.11, in: Proceedings of the 7th International Conference on Mobile Computing And Networking (MobiCom'01), July, 2001.
15. W. Diffie and M.E. Hellman: New Directions in cryptography. IEEE Transactions on Information Theory, vol.22, pp. 644-654, 1976.
16. W. Diffie, P. van Oorschot, and M. Wiener: Authentication and authenticated key exchanges, Designs, Codes, and Cryptography, vol. 2, (Kluwer Academic Publishers) pp. 107–125, 1992.
17. T. ElGamal: A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions on Information Theory, vol.31, no.4, pp. 469–472, 1985.
18. 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.
19. I. Ingemarsson, D. Tang, and C. Wong: A conference key distribution system, IEEE Transactions on Information Theory, vol.28, no.5, pp. 714–720, 1982.
20. M. Just and S. Vaudenay: Authenticated multi-party key agreement, in: Advances in Cryptology – Asiacrypt'96, LNCS 1163, pp. 36–49, 1996.
21. Y. Kim, A. Perrig, and G. Tsudik: Simple and fault-tolerant key agreement for dynamic collaborative groups, in: Proceedings of 1st ACM Conference on Computer and Communications Security (CCS'00), pp. 235–244, 2000.
22. Y. Kim, A. Perrig, and G. Tsudik: Communication-efficient group key agreement, in: Proceedings of International Federation for Information Processing (IFIP SEC'01), pp. 229–244, 2001.
23. J. Katz and M. Yung: Scalable protocols for authenticated group key exchange, in: Advances in Cryptology – Crypto'03, LNCS 2729, pp. 110–125, 2003.
24. A. Shamir: How to share a secret, Communications of the ACM, vol.22, no.11, pp. 612–613, 1979.

25. D.G. Steer, L. Strawczynski, W. Diffie, and M. Wiener: A secure audio teleconference system, in: Advances in Cryptology – Crypto'88, LNCS 403, pp. 520–528, 1988.
26. M. Steiner, G. Tsudik, and M. Waidner: Key agreement in dynamic peer groups, IEEE Transactions on Parallel and Distributed Systems, vol.11, no.8, pp. 769–780, August 2000.
27. H.-T. Yeh and H.-M. Sun: Password-based user authentication and key distribution protocols for client-server applications, The Journal of Systems and Software, vol.72, no.1, pp. 97–103, 2004.