Security Improvement on a Group Key Exchange Protocol for Mobile Networks*

Junghyun Nam[1], Kwangwoo Lee[2], Juryon Paik[2], Woojin Paik[1], and Dongho Won[2,**]

[1] Department of Computer Engineering, Konkuk University, Korea
jhnam@kku.ac.kr,wjpaik@kku.ac.kr
[2] Department of Computer Engineering, Sungkyunkwan University, Korea
kwlee@security.re.kr,wise96@ece.skku.ac.kr,dhwon@security.re.kr

Abstract. A group key exchange (GKE) protocol is designed to allow a group of parties communicating over a public network to establish a common secret key called a *session key*. As group-oriented applications gain popularity over the Internet, a number of GKE protocols have been suggested to provide those applications with a secure multicast channel. Among the many protocols is the GKE protocol presented by Dutta and Dowling for mobile ad hoc networks. In this paper, we are concerned with the security of the Dutta-Dowling protocol. Their protocol carries a proof of security in the standard adversarial model which captures unknown key-share attacks. But unlike the claim of provable security, the Dutta-Dowling protocol fails to achieve unknown key-share resilience. We here reveal this security vulnerability of the protocol and show how to address it.

Keywords: Mobile ad hoc network, group key exchange, cluster-based network, unknown key-share attack.

1 Introduction

Mobile ad hoc networks have matured as a viable means to provide anytime-anywhere networking services for infrastructure-free communication over a shared wireless channel. With the broad range of wirelessly connected mobile devices being used today, security in mobile ad hoc networks is of essential and increasing importance. However, it still remains a difficult task to provide strong protection for mobile ad hoc networks where nodes may be resource constrained. Although mobile computing technology has become more powerful and accessible than ever before, mobile devices are typically characterized by low processing capability and limited power supply [14] which are inherent to the mobility nature. It is thus necessary that the cost due to security-related operations should be

* This work was supported by Priority Research Centers Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0020210).
** Corresponding author.

minimized for mobile devices in such a way that the required security goals are not compromised. This requirement makes it especially difficult to secure mobile ad hoc networks. In fact despite all the research efforts made so far, security is still a major limiting factor towards the eventual success of mobile ad hoc networks [2,8,13,12].

Recently, Dutta and Dowling [7] proposed a group key exchange (GKE) protocol for securing mobile ad hoc networks. The Dutta-Dowling protocol divides the whole nodes in a mobile ad hoc network into a set of clusters. Each cluster consists of a single sponsor node and multiple regular nodes. A sponsor node, as the representative of its cluster, is responsible for communicating with the sponsors of other clusters. Such cluster-based networks can be found in many applications including military missions and environmental monitoring. For instance, a military network consists of mobile devices carried by soldiers, automatic weapons, sensing devices, etc. In this setting, a platoon commander may act as a sponsor of its cluster and may be able to communicate with platoon commanders of other clusters.

The GKE protocol presented by Dutta and Dowling was claimed to be provably secure in a formal security model which captures unknown key-share attacks. But despite the claim of provable security, their GKE protocol is in fact insecure against an unknown key-share attack. In the current paper, we report this security problem with the Dutta-Dowling protocol and figure out how to solve it. Our result implies that the claimed proof of security for the Dutta-Dowling protocol is invalid.

The remainder of this paper is organized as follows. Section 2 reviews the GKE protocol proposed by Dutta and Dowling. Section 3 presents our known key-share attack on the Dutta-Dowling protocol and offers a security patch for the protocol. Section 4 provides an overview of the formal proof model for the protocol, along with an associated definition of security. Section 5 breaks the so-called "AKE security" of the protocol.

2 The Dutta-Dowling Protocol

Dutta and Dowling [7] construct their protocol in two steps: first, they present an unauthenticated GKE protocol UP; then, they transform UP into an authenticated protocol AP. The transformation from UP to AP is done by using a simplified variant of the compiler presented by Katz and Yung [11].

2.1 Unauthenticated Protocol UP

Let $\mathbb{G}$ be be a cyclic (multiplicative) group of prime order q, and let g be a random fixed generator of $\mathbb{G}$. Let $H : \mathbb{G} \rightarrow \mathbb{Z}_q^*$ be a cryptographically secure hash function. The protocol UP is designed to be secure against passive adversaries. The users participating in UP are divided into into small groups called clusters. Assume there are n clusters $C_1, \ldots, C_n$. Let S_i denote the sponsor of cluster C_i. UP runs in two phases:

Phase 1 (Cluster Key Generation): The users in each C_i generate a cluster key CK_i by running the protocol Setup (described below).

Phase 2 (Group Key Generation): Once all cluster keys CK_1, …, CK_n have been generated, the sponsors $S_1, \ldots, S_n$ run the Setup protocol again to establish a group session key SK. But this time, each S_i sets its random exponent x_i equal to $H(CK_i)$ so that other users in C_i can compute the session key. In this run of Setup, it is assumed that the broadcast messages are received not only by sponsors but also by all other users.

In the following description of protocol Setup, all indices are to be taken in a cycle, i.e., $U_{n+1} = U_1$, etc.

Protocol Setup

Round 1: Each user U_i chooses a random $x_i \in \mathbb{Z}_q^*$ and broadcasts $y_i = g^{x_i}$.

Round 2: Each user U_i computes the left key $K_i^L = y_{i-1}^{x_i}$ and the right key $K_i^R = y_{i+1}^{x_i}$ and broadcasts $Y_i = K_i^R / K_i^L$.

Key Computation: Each U_i computes

$$\begin{aligned} K_{i+1}^R &= Y_{i+1} K_i^R, \\ K_{i+2}^R &= Y_{i+2} K_{i+1}^R, \\ &\vdots \\ K_{i+n-1}^R &= Y_{i+n-1} K_{i+n-2}^R. \end{aligned}$$

U_i verifies that K_{i+n-1}^R is equal to K_i^L. U_i aborts if the verification fails. Otherwise, U_i computes the session key SK as:

$$\begin{aligned} SK &= K_1^R K_2^R \cdots K_n^R \\ &= g^{x_1 x_2 + x_2 x_3 + \cdots + x_n x_1}. \end{aligned}$$

2.2 Authenticated Protocol AP

Let $\Sigma = (\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme which is strongly unforgeable under adaptive chosen message attack. Here, Kgen is the key generation algorithm, Sign is the signature generation algorithm, and Vrfy is the signature verification algorithm. During the initialization phase of AP, each user U_i generates its long-term verification/signing keys (PK_i, SK_i) by running $\mathsf{Kgen}(1^\kappa)$ and makes the verification key PK_i public. Each user U_i should maintain a counter c_i to participate in the protocol AP. The counter c_i is incremented every time U_i participates in a new run of AP. Let $U_1, \ldots, U_n$ be the users who wish to establish a common session key. As part of the initialization, each U_i sets $\mathsf{pid}_i = \{U_1, \ldots, U_n\}$. In the protocol AP, the users simply perform the protocol UP, but signing outgoing messages and verifying the correctness of incoming messages as follows:

– Let m_i^t be the t-th message that U_i is supposed to broadcast as part of protocol UP. Then, U_i computes the signature $\sigma_i^t = \mathsf{Sign}_{SK_i}(U_i|t|m_i^t|c_i)$ and broadcasts $M_i^t = U_i|t|m_i^t|c_i|\sigma_i^t$ as the replacement of m_i^t.
– When U_i receives message $M_j^t = U_j|t|m_j^t|c_j|\sigma_j^t$ from U_j, it checks that: (1) $U_j \in \mathsf{pid}_i$, (2) t is the next expected sequence number for messages from U_j, and (3) $\mathsf{Vrfy}_{PK_j}(U_j|t|m_j^t|c_j, \sigma_j^t) = 1$. U_i aborts if any of these conditions fail to hold. Otherwise, U_i continues as it would in UP upon receiving message M_j^t.
– Each non-aborted user U_i computes the session key as in UP and defines the session identifier $\mathsf{sid}_i = \langle (U_1, c_1), \dots, (U_n, c_n) \rangle$.

The above procedure for converting UP into AP is a variant of the Katz-Yung compiler [11] which transforms any GKE protocol secure against a passive adversary into one that is secure against an active adversary. But unlike the Katz-Yung compiler, the transformation here makes use of counters instead of random nonces and accordingly does not require an additional round for exchanging random nonces among users.

3 Security Analysis

This section conducts a security analysis on the above-described protocol AP. We first reveal the vulnerability of AP to an unknown key-share attack and then figure out how to fix the protocol.

3.1 Unknown Key-Share Attack

The protocol AP is vulnerable to an unknown key-share attack in the presence of an active adversary. An adversary U_A is said to succeed in an unknown key-share attack if at the end of the attack, there exist two parties U_i and U_j such that: *(1) U_i and U_j have computed the same key; (2) U_i is unaware of the "key share" with U_j and mistakenly believes that its key is shared with U_A; and (3) U_j correctly believes that its key is shared with U_i.* As implied by this definition, U_A need not obtain any session key to benefit from an unknown key-share attack [6,1,9,4]. Consider a protocol session S to be run by the users of group $\mathcal{G} = \{U_1, U_2, U_3\}$. Now suppose that U_1 and U_2 accept the invitation by U_A to participate in a new concurrent session S', thus forming the group $\mathcal{G}' = \{U_1, U_2, U_A\}$. We will consider the users of two sessions as sponsors who have already established their respective cluster keys and yet need to establish a common group key by running the protocol Setup. Hereafter, we use Π_i and Π_i' to denote U_i's instances participating respectively in S and S'. Let $\textsc{Exp}_i$ and $\textsc{Div}_i$ be the first and second messages sent by U_i in the authenticated version of Setup. Let α_i and β_i be the signatures contained in $\textsc{Exp}_i$ and $\textsc{Div}_i$. The attack works as follows:

1. As session S starts, Π_1, Π_2 and U_3 will send their first messages $\textsc{Exp}_1 = U_1|1|y_1|c_1|\alpha_1$, $\textsc{Exp}_2 = U_2|1|y_2|c_2|\alpha_2$ and $\textsc{Exp}_3 = U_3|1|y_3|c_3|\alpha_3$, respectively. The adversary U_A intercepts $\textsc{Exp}_3$ while blocking $\textsc{Exp}_1$ and $\textsc{Exp}_2$ from

reaching U_3. In other words, Π_1 and Π_2 never receive $\textsc{Exp}_3$ and U_3 receives neither $\textsc{Exp}_1$ nor $\textsc{Exp}_2$.

2. As a participant of session S', the adversary U_A sends the message $\textsc{Exp}_A = U_A|1|y_3|c_A|\alpha_A$ to Π'_1 and Π'_2 and receives the messages $\textsc{Exp}'_1 = U_1|1|y'_1|c'_1|\alpha'_1$ and $\textsc{Exp}'_2 = U_2|1|y'_2|c'_2|\alpha'_2$ respectively from Π'_1 and Π'_2. Notice that $\textsc{Exp}_A$ contains y_3 (which is generated by U_3 and is obtainable from $\textsc{Exp}_3$). We mean by this that U_A has computed its signature α_A as $\alpha_A = \mathsf{Sign}_{SK_A}(U_A|1|y_3|c_A)$.
3. U_A forwards the received messages $\textsc{Exp}'_1$ and $\textsc{Exp}'_2$ to U_3 as if they are sent by Π_1 and Π_2, respectively. These messages will pass U_3's verification since α'_1 (resp. α'_2) is a valid signature on $U_1|1|y'_1|c'_1$ (resp. $U_2|1|y'_2|c'_2$) under the verification key PK_1 (resp. PK_2). Hence, U_3 will send out its second message $\textsc{Div}_3 = U_3|2|Y_3|c_3|\beta_3$. If we let $y'_1 = g^{x'_1}$ and $y'_2 = g^{x'_2}$, then clearly
$$Y_3 = g^{x_3x'_1 - x'_2x_3}.$$
4. U_A intercepts $\textsc{Div}_3$, computes $\beta_A = \mathsf{Sign}_{SK_A}(U_A|2|Y_3|c_A)$ (using Y_3 from U_3), and sends $\textsc{Div}_A = U_A|2|Y_3|c_A|\beta_A$ to Π'_1 and Π'_2. Meanwhile, Π'_1 and Π'_2 will send U_A their respective messages $\textsc{Div}'_1 = U_1|2|Y'_1|c'_1|\beta'_1$ and $\textsc{Div}'_2 = U_2|2|Y'_2|\ c'_2|\beta'_2$ where
$$Y'_1 = g^{x'_1x'_2 - x_3x'_1} \text{ and } Y'_2 = g^{x'_2x_3 - x'_1x'_2}.$$
5. U_A forwards $\textsc{Div}'_1$ and $\textsc{Div}'_2$ to U_3 as if they are from Π_1 and Π_2, respectively. These messages will pass U_3's verifications since the signatures β'_1 and β'_2 are both valid and the following equation holds: $K_3^L = Y'_2Y'_1K_3^R$, where $K_3^L = g^{x'_2x_3}$ and $K_3^R = g^{x_3x'_1}$.
6. Consequently, Π'_1, Π'_2 and U_3 will compute the same session key $SK = g^{x'_1x'_2 + x'_2x_3 + x_3x'_1}$.

At the end of the attack: (1) U_1, U_2 and U_3 have computed the same session key SK; (2) U_1 and U_2 believe that SK is shared with U_A, while in fact it is shared with U_3; (3) U_3 believes that SK is shared with U_1 and U_2. This shows that the protocol AP is vulnerable to an unknown key-share attack when two protocol sessions are running concurrently with some joint participants.

3.2 Countermeasure

The vulnerability of AP to the unknown key-share attack stems from the fact that U_3 cannot distinguish between the signatures generated by Π_1 (resp. Π_2) and those generated by Π'_1 (resp. Π'_2). A simple way to fix the protocol is to change the computations of the signatures α_i and β_i to $\alpha_i = \mathsf{Sign}_{SK_i}(U_i|1|y_i|c_i|\mathsf{pid}_i)$ and $\beta_i = \mathsf{Sign}_{SK_i}(U_i|2|Y_i|c_i|\mathsf{pid}_i)$. α_i and β_i are now computed by including participants' identities as part of the messages being signed. With this modification applied, the signatures from Π'_1 and Π'_2 can no longer pass U_3's verification since the participants are different between the two sessions S and S'. Hence, the unknown key-share attack is not valid against the improved protocol.

4 Overview of Security Model

The protocol AP comes along with a claimed proof of its security in a formal model of communication and adversarial capabilities. The proof model used for AP is based on Bresson et al.'s 2002 model for dynamic group key exchange [3]. Here we break the security of AP in the context of the proof model.

4.1 Participants

Let $\mathcal{U}$ be a set of all users who can participate in a group key exchange protocol. The users in any subset of $\mathcal{U}$ may run the group key exchange protocol at any point in time to establish a session key. Each user may run the protocol multiple times either serially or concurrently, with possibly different groups of participants. Thus, at a given time, there could be many instances of a single user. We use Π_i^π to denote the π-th instance of user U_i. Before the protocol is executed for the first time, each user $U_i \in \mathcal{U}$ creates a long-term public/private key pair (PK_i, SK_i) by running a key generation algorithm $\mathcal{K}(1^\kappa)$. All instances of a user share the public/private keys of the user even if they participate in their respective sessions independently. Each private key is kept secret by its owner while the public keys of all users are publicized.

4.2 Adversary

The adversary is in complete control of every aspect of all communications between participants, and may ask, at any time, them to open up access to their long-term secret keys. These capabilities and others of the adversary are modeled via various oracles to which the adversary is allowed to make queries.

- $\mathsf{Send}(\Pi_i^\pi, m)$: This query sends message m to instance Π_i^π. The instance Π_i^π proceeds as it would in the protocol upon receiving m; the instance updates its state performing any required computation, and generates and sends out a response message as needed. The response message, if any, is the output of this query and is returned to the adversary. This models active attacks on the protocol, allowing the adversary to control at will all message flows between instances. A query of the form $\mathsf{Send}(\Pi_i^\pi, \langle U_1, \ldots, U_n \rangle)$ prompts Π_i^π to initiate a protocol run in which the participants are $U_1, \ldots, U_n \in \mathcal{U}$.
- $\mathsf{Execute}(U_1, \ldots, U_n)$: This query lets the protocol be executed among the users $U_1, \ldots, U_n$ (or rather among their instances). The transcript of the protocol execution is returned to the adversary as the output of the query. This oracle call represents passive eavesdropping of a protocol execution.
- $\mathsf{Reveal}(\Pi_i^\pi)$: This query returns to the adversary the session key SK_i^π held by instance Π_i^π. This oracle call captures the idea that exposure of some session keys should not affect the security of other session keys.
- $\mathsf{Corrupt}(U_i)$: This query returns all long-term secrets of U_i. The adversary can issue this query at any time regardless of whether U_i is currently executing the protocol or not. This oracle call captures the idea that damage due to

loss of U_i's long-term secrets should be restricted to those sessions where U_i will participate in the future.

– $\mathsf{Test}(\Pi_i^\pi)$: This oracle call does not model any capability of the adversary, but is used to define the semantic security of the session key SK_i^π. The output of this query depends on the hidden bit $\mathbf{b}$ chosen uniformly at random from $\{0,1\}$. The Test oracle returns the real session key held by Π_i^π if $\mathbf{b} = 1$, or returns a random key drawn from the session-key space if $\mathbf{b} = 0$. The adversary is allowed to access the Test oracle only once, and the only Test query must be directed to a *fresh* instance (see Section 4.3 for the definition of freshness).

Remark 1. A common misunderstanding regarding the Corrupt oracle is that it is included in the model only for the purpose of dealing with (perfect) forward secrecy. The truth is that Corrupt queries not only model forward secrecy but also capture a variety of impersonation attacks. For example, key exchange protocols proven secure in a model that allows Corrupt queries ought to be secure against unknown key share attacks [4]. Another example is attacks by malicious insiders, though we focus on outsider attacks in this work. We refer the interested reader to [10] for details on how Corrupt queries are used to model insider attacks. Basically, the corruption of a principle U_i should not lead the adversary to have the ability to impersonate any principle other than U_i, because such ability would endanger even those sessions where U_i is not invited to participate.

4.3 Security Definition

Definition of security of a GKE protocol is based on the notion of *freshness* which in turn is defined using the notion of *partnership.*

Partnership. Like most of previous works, we define partnership between instances via *session IDs* and *partner IDs.* Literally, a session ID is a unique identifier of a protocol session. A partner ID is the identities of a group of users who intend to establish their common session key. Session IDs are typically computed during protocol runs whereas partner IDs should be given as input to protocol participants. We let sid_i^π be the session ID computed by instance Π_i^π and let pid_i^π be the partner ID provided to instance Π_i^π. An instance is said to *accept* when it successfully computes a session key in a protocol execution. Let acc_i^π be a boolean variable that evaluates to TRUE if Π_i^π has accepted, and FALSE otherwise. We say that any two instances Π_i^π and Π_j^ω are *partnered* if all the following three conditions are satisfied: (1) $\mathsf{sid}_i^\pi = \mathsf{sid}_j^\omega$, (2) $\mathsf{pid}_i^\pi = \mathsf{pid}_j^\omega$, and (3) $\mathsf{acc}_i^\pi = \mathsf{acc}_j^\omega = \text{TRUE}$.

Freshness. An unfresh instance is an instance whose session key can be exposed by trivial means. More precisely:

Definition 1. *The instance Π_i^π is unfresh if any of the following conditions holds:*

1. *The adversary queried* $\mathsf{Reveal}(\Pi_i^\pi)$ *or* $\mathsf{Reveal}(\Pi_j^\omega)$ *where Π_i^π and Π_j^ω are partnered.*

2. *A* $\mathsf{Corrupt}$ *query was asked for some user in* pid_i^π *before a* Send *query has been asked for an instance of some user in* pid_i^π*.*

All other instances are considered fresh.

As already mentioned, the adversary is disallowed to make a Test query against an unfresh instance.

Security. The security of a GKE protocol P against an adversary $\mathcal{A}$ is defined in terms of the probability that $\mathcal{A}$ succeeds in distinguishing a real session key established in an execution of P from a random session key. That is, the adversary $\mathcal{A}$ is considered successful in attacking P if it breaks the semantic security of session keys generated by P. More precisely, the security is defined in the following context. The adversary $\mathcal{A}$ executes the protocol exploiting as much parallelism as possible and asking any queries allowed in the model. During executions of the protocol, the adversary $\mathcal{A}$, at any time, asks a Test query to a fresh instance, gets back a key as the response to this query, and at some later point in time, outputs a bit $\mathbf{b}'$ as a guess for the value of the hidden bit $\mathbf{b}$ used by the Test oracle.[1] Then the advantage of $\mathcal{A}$ in attacking protocol P is denoted by $\mathsf{Adv}_{\mathsf{P}}(\mathcal{A})$, and is defined as

$$\mathsf{Adv}_{\mathsf{P}}(\mathcal{A}) = |2 \cdot \Pr[\mathbf{b} = \mathbf{b}'] - 1|.$$

This notion of security is commonly termed as "AKE security". Protocol P is said to be *AKE-secure* if the advantage $\mathsf{Adv}_{\mathsf{P}}(\mathcal{A})$ is negligible (as a function of the security parameter) for all probabilistic polynomial time adversaries $\mathcal{A}$.

5 Breaking AKE Security

AKE-secure protocols should be secure against unknown key-share attacks. The protocol AP carries a claimed proof of its AKE security, but as we have seen, it is not secure against an unknown key-share attack. This implies that the security proof for AP is flawed. In this section, we interpret our attack in the context of the formal proof model to show that the attack does indeed break the AKE security of the protocol.

Our attack on AP is well captured in the proof model. Table 1 shows the sequence of oracle queries corresponding to the attack scenario given in Section 3.1. The goal of the adversary $\mathcal{A}$ is to break the AKE security of AP. $\mathcal{A}$ begins by asking $\mathsf{Corrupt}(U_A)$ to obtain the signing key SK_A of U_A. Then $\mathcal{A}$ initiates two sessions $S : \{U_1, U_2, U_3\}$ and $S' : \{U_1, U_2, U_A\}$ by asking the initial Send queries: $\mathsf{Send}(\Pi_1^1, \langle U_1, U_2, U_3\rangle)$, $\mathsf{Send}(\Pi_2^1, \langle U_1, U_2, U_3\rangle)$, $\mathsf{Send}(\Pi_3^1, \langle U_1, U_2, U_3\rangle)$, $\mathsf{Send}(\Pi_1^2, \langle U_1, U_2, U_A\rangle)$, $\mathsf{Send}(\Pi_2^2, \langle U_1, U_2, U_A\rangle)$. Notice that no instance of U_A needs to be asked this form of Send query because $\mathcal{A}$ will simulate by itself the actions of U_A. The rest of the queries are straightforward from the attack scenario.

[1] Of course, the adversary is prohibited from asking a Reveal query to the instance that is partnered with the tested instance, because such Reveal query would trivially expose the test session key.

Table 1. The sequence of oracle queries corresponding to the unknown key-share attack described in Section 3.1

	Query	Response
1	$\mathsf{Corrupt}(U_A)$	SK_A
2	$\mathsf{Send}(\Pi_1^1, \langle U_1, U_2, U_3 \rangle)$	$\text{Exp}_1 = U_1\|1\|y_1\|c_1\|\alpha_1$
3	$\mathsf{Send}(\Pi_2^1, \langle U_1, U_2, U_3 \rangle)$	$\text{Exp}_2 = U_2\|1\|y_2\|c_2\|\alpha_2$
4	$\mathsf{Send}(\Pi_3^1, \langle U_1, U_2, U_3 \rangle)$	$\text{Exp}_3 = U_3\|1\|y_3\|c_3\|\alpha_3$
5	$\mathsf{Send}(\Pi_1^2, \langle U_1, U_2, U_A \rangle)$	$\text{Exp}'_1 = U_1\|1\|y'_1\|c'_1\|\alpha'_1$
6	$\mathsf{Send}(\Pi_2^2, \langle U_1, U_2, U_A \rangle)$	$\text{Exp}'_2 = U_2\|1\|y'_2\|c'_2\|\alpha'_2$
7	$\mathsf{Send}(\Pi_1^2, \text{Exp}_A = U_A\|1\|y_3\|c_A\|\alpha_A)$	
8	$\mathsf{Send}(\Pi_1^2, \text{Exp}'_2 = U_2\|1\|y'_2\|c'_2\|\alpha'_2)$	$\text{Div}'_1 = U_1\|2\|Y'_1\|c'_1\|\beta'_1$
9	$\mathsf{Send}(\Pi_2^2, \text{Exp}_A = U_A\|1\|y_3\|c_A\|\alpha_A)$	
10	$\mathsf{Send}(\Pi_2^2, \text{Exp}'_1 = U_1\|1\|y'_1\|c'_1\|\alpha'_1)$	$\text{Div}'_2 = U_2\|2\|Y'_2\|c'_2\|\beta'_2$
11	$\mathsf{Send}(\Pi_3^1, \text{Exp}'_1 = U_1\|1\|y'_1\|c'_1\|\alpha'_1)$	
12	$\mathsf{Send}(\Pi_3^1, \text{Exp}'_2 = U_2\|1\|y'_2\|c'_2\|\alpha'_2)$	$\text{Div}_3 = U_3\|2\|Y_3\|c_3\|\beta_3$
13	$\mathsf{Send}(\Pi_1^2, \text{Div}_A = U_A\|2\|Y_3\|c_A\|\beta_A)$	
14	$\mathsf{Send}(\Pi_1^2, \text{Div}'_2 = U_2\|2\|Y'_2\|\ c'_2\|\beta'_2)$	(accept)
15	$\mathsf{Send}(\Pi_2^2, \text{Div}_A = U_A\|2\|Y_3\|c_A\|\beta_A)$	
16	$\mathsf{Send}(\Pi_2^2, \text{Div}'_1 = U_1\|2\|Y'_1\|c'_1\|\beta'_1)$	(accept)
17	$\mathsf{Send}(\Pi_3^1, \text{Div}'_1 = U_1\|2\|Y'_1\|c'_1\|\beta'_1)$	
18	$\mathsf{Send}(\Pi_3^1, \text{Div}'_2 = U_2\|2\|Y'_2\|\ c'_2\|\beta'_2)$	(accept)
19	$\mathsf{Reveal}(\Pi_1^2)$ (or $\mathsf{Reveal}(\Pi_2^2)$)	SK_1^2 (or SK_2^2)
20	$\mathsf{Test}(\Pi_3^1)$	$\overline{SK}$

$\overline{SK}$: either the real session key SK_3^1 or a random session key.

The instance Π_3^1 accepts when the 18-th query is asked while Π_1^2 (resp. Π_2^2) accepts when the 14-th (resp. 16-th) query is asked. The computed session keys SK_1^2, SK_2^2 and SK_3^1 are all set equal to $g^{x'_1x'_2+x'_2x_3+x_3x'_1}$. The instance Π_3^1 is fresh under Definition 1; no one in $\mathsf{pid}_3^1 = \{U_1, U_2, U_3\}$ has been sent a $\mathsf{Corrupt}$ query and no Reveal query has been made against Π_3^1 or its partners. (Notice that by definition, Π_3^1 is not partnered with Π_1^2 and Π_2^2 since pid_3^1 is different from pid_1^2 and pid_2^2.) Thus, $\mathcal{A}$ may test (i.e., ask the Test query against) the instance Π_3^1. Because $\mathcal{A}$ may query $\mathsf{Reveal}(\Pi_1^2)$ and $\mathsf{Reveal}(\Pi_2^2)$ without affecting the freshness of Π_3^1, it follows that $\Pr[\mathbf{b} = \mathbf{b}'] = 1$ and hence $\mathsf{Adv}_{\mathsf{AP}}(\mathcal{A}) = 1$. Therefore, $\mathcal{A}$ achieves its goal of breaking the AKE security of protocol AP.

References

1. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the station-to-station (STS) protocol. In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 154–170. Springer, Heidelberg (1999)
2. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: the insecurity of 802.11. In: 7th ACM Conference on Mobile Computing and Networking, pp. 180–189 (2001)
3. Bresson, E., Chevassut, O., Pointcheval, D.: Dynamic group Diffie-Hellman key exchange under standard assumptions. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 321–336. Springer, Heidelberg (2002)

4. Choo, K.-K., Boyd, C., Hitchcock, Y.: Errors in computational complexity proofs for protocols. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 624–643. Springer, Heidelberg (2005)
5. Choo, K.-K.R., Boyd, C., Hitchcock, Y., Maitland, G.: On session identifiers in provably secure protocols. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 351–366. Springer, Heidelberg (2005)
6. Diffie, W., Oorschot, P., Wiener, M.: Authentication and authenticated key exchanges. Designs, Codes, and Cryptography 2(2), 107–125 (1992)
7. Dutta, R., Dowling, T.: Secure and efficient group key agreements for cluster based networks. In: Gavrilova, M.L., Tan, C.J.K., Moreno, E.D. (eds.) Transactions on Computational Science IV. LNCS, vol. 5430, pp. 87–116. Springer, Heidelberg (2009)
8. Johnston, D., Walker, J.: Overview of IEEE 802.16 security. IEEE Security and Privacy Magazine 2(3), 40–48 (2004)
9. Kaliski, B.S.: An unknown key-share attack on the MQV key agreement protocol. ACM Transactions on Information and System Security 4(3), 275–288 (2001)
10. Katz, J., Shin, J.: Modeling insider attacks on group key-exchange protocols. In: 12th ACM Conference on Computer and Communications Security (CCS 2005), pp. 180–189 (2005)
11. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 110–125. Springer, Heidelberg (2003)
12. Nam, J., Kim, S., Won, D.: A weakness in the Bresson-Chevassut-Essiari-Pointcheval's group key agreement scheme for low-power mobile devices. IEEE Communications Letters 9(5), 429–431 (2005)
13. Ng, S.-L., Mitchell, C.: Comments on mutual authentication and key exchange protocols for low power wireless communications. IEEE Communications Letters 8(4), 262–263 (2004)
14. Potlapally, N.R., Ravi, S., Raghunathan, A., Jha, N.K.: Analyzing the energy consumption of security protocols. In: 2003 ACM International Symposium on Low Power Electronics and Design, pp. 30–35 (2003)