A Security Weakness in a Generic Construction of a Group Key Exchange Protocol

Junghyun Nam[†]

April 22, 2010

Abstract

Protocols for group key exchange are cryptographic algorithms that allow a group of parties communicating over a public network to come up with a common secret key. One of the interesting results of research on group key exchange is the protocol compiler presented by Abdalla et al. in TCC '07. Abdalla et al.'s compiler shows how one can transform any authenticated 2-party key exchange protocol into an authenticated group key exchange protocol with 2 more rounds of communication. This compiler certainly is elegant in its genericness, symmetricity, simplicity and efficiency. However, the situation completely changes when it comes to security. In this work, we reveal a major security weakness in Abdalla et al.'s compiler and show how to address it. The security weakness uncovered here implies that Abdalla et al.'s proof of security for their compiler is invalid.

Keywords: Cryptography, Group key exchange, Protocol compiler, Implicit key authentication, Key confirmation.

1 Introduction

The primary goal of cryptography is to provide a means for communicating confidentially and with integrity over a public channel. In practice, this goal is often achieved with key exchange protocols which allow the parties communicating over an insecure network to establish a common secret key called a *session key*. Typically, the communicating parties, who want confidentiality and integrity, first generate a session key by running an appropriate key exchange protocol and then use this key together with standard cryptographic algorithms for message encryption and authentication. Thus, the problem of establishing confidential and integrity-preserving communication is commonly reduced to the problem of getting a right protocol for session key generation. Needless to say, the initial work of Diffie and Hellman [2] has been followed by a

[†]Department of Computer Science, Konkuk University, 322 Danwol-dong, Chungju-si, Chungcheongbuk-do 380-701, Republic of Korea.
E-mail: jhnam@kku.ac.kr

tremendous amount of research effort aimed at designing and analyzing key exchange protocols.

The highest priority in designing a key exchange protocol is placed on the security of session keys to be established by the protocol. Roughly speaking, establishing a session key securely means that the key is being known only to the intended parties at the end of the protocol run. Even if it is computationally infeasible to break the cryptographic algorithms used, the whole system becomes vulnerable to all manner of attacks if the keys are not securely established. But unfortunately, the experience has shown that the design of secure key exchange protocols is notoriously difficult. In particular, the difficulty is greatly increased in the group setting where a session key is to be established among an arbitrary number of parties. Indeed, there is a long history of protocols for this domain being proposed and years later found to be flawed (e.g., [5, 6, 4]). Thus, group key exchange protocols must be subjected to a thorough and systematic scrutiny before they are deployed into a public network, which might be controlled by an adversary.

The complexity of designing a group key exchange protocol has prompted modular approaches where the whole design process is broken down into simpler, more manageable steps. One such approach is 2-to-n protocol compilers, which assume an arbitrary 2-party key exchange protocol and use it as a key component in building a group key exchange protocol. 2-to-n compilers are certainly useful in that an n-party solution can be generically constructed from any existing 2-party solution. Of course, such generic construction may become meaningless if the compiler is not secure. A secure compiler would mean, informally, that it yields a secure n-party solution as long as the given 2-party solution is secure.

In 2007, Abdalla et al. [1] presented an interesting 2-to-n compiler. One of the attractive features of Abdalla et al.'s compiler is that it requires no further long-term secrets for authentication than those used in the underlying 2-party protocol. This feature implies that if the given 2-party protocol is password-only authenticated, then the group key exchange protocol output by the compiler is password-only authenticated as well. Moreover, Abdalla et al.'s compiler is quite efficient in terms of both computation and communication costs. Applying the compiler to a 2-party protocol increases the amount of computation per user by a factor of $O(n)$ and the number of communication rounds only by a factor of 2. From a practical point of view, the $O(n)$ increase of computation overhead is not that significant because the increase is purely due to exclusive-or (XOR) operations, which can be implemented efficiently in hardware and/or software. However, we found that Abdalla et al.'s compiler is not satisfactory with respect to security. Despite the claim of provable security, the group key exchange protocol constructed by the compiler from a 2-party protocol exhibits insecurity in the face of an active adversary. In this work, we report this security problem with Abdalla et al.'s compiler and figure out how to solve it. As will be discussed in Section 3, our result invalidates Abdalla et al.'s proof of security for the compiler.

2 Abdalla et al.'s Group Key Exchange

This section reviews Abdalla et al.'s compiler [1] that aims to construct an authenticated group key exchange protocol GKEP from any authenticated 2-party key exchange protocol 2KEP. Let $\mathcal{U}$ be a polynomial-sized set of all users who are potentially interested in participating in the protocol GKEP. The users in any subset of $\mathcal{U}$ may run GKEP at any point in time to establish a common session key. A user $U_i \in \mathcal{U}$ may have several instances involved in distinct, possibly concurrent, executions of GKEP. The network where the users interact is fully controlled by an active adversary who may read, intercept, delay and fabricate any messages at will.

During a trusted initialization phase which occurs before GKEP is ever executed, the users in $\mathcal{U}$ generate their long-term (low- or high-entropy) secrets needed for authentication. Those long-term secrets are solely for use in 2KEP because the transformation itself, from 2KEP to GKEP, requires no long-term secrets. Thus, any initial setup assumptions for authentication of 2-party key exchange can be made for the authentication of GKEP, including the following typical ones:

- Authentication is based on public-key cryptography: each user $U_i \in \mathcal{U}$ owns a pair of private/public (or, signing/verification) keys. The private key is kept secret while the public key is made publicly available.
- Authentication is based on symmetric cryptography: each pair of users $U_i, U_j \in \mathcal{U}$ shares a high-entropy symmetric key.
- Authentication is based on passwords: each pair of users $U_i, U_j \in \mathcal{U}$ shares a low-entropy password.

If authentication is based on shared secrets, GKEP also works in the setting where the complete set of protocol participants shares one common secret. But we exclude this case from the consideration since our security analysis on GKEP does not hold for the case (for more discussion on this point, see Remark 1 at the end of the next section). We instead assume that any one of the three typical setups listed above is configured in the initialization phase of GKEP. But this assumption is not necessary for our result and is made only for clarity. Indeed, as will be apparent in the next section, our result still holds under the weaker assumption that all concurrent instances of a user use the same long-term secret(s).

The cryptographic tools used in the construction of GKEP are:

- **a collision-resistant pseudorandom function family** $\mathcal{F} = \{F^\ell\}_{\ell \in \mathbb{N}}$ with $F^\ell = \{F^\ell_s\}_{s \in \{0,1\}^L}$. Informally, collision-resistance means that there exists a value v such that no efficient adversary can find two different indices $s, s' \in \{0,1\}^L$ such that $F_s(v) = F_{s'}(v)$. The compiler assumes two publicly known values v_0 and v_1 that satisfies the collision-resistance condition.

- **a hash function** H selected from a family of universal hash functions. H outputs an L-bit string and is used to select an index within the aforementioned collision-resistant pseudorandom function family.
- **a non-interactive non-malleable commitment scheme** C that satisfies the following requirements:
  1. it must be *perfectly binding*, i.e., every commitment c defines at most one value $\mathsf{decommit}(c)$;
  2. it must achieve *non-malleability for multiple commitments* — if an adversary receives commitments to a (polynomial sized) set of values ν, she must not be able to output commitments to a (polynomial sized) set of values μ related to ν in a known way.

The underlying 2-party protocol $\mathsf{2KEP}$, upon input of two users $U_i, U_j \in \mathcal{U}$ (or rather their identities), returns either a secret key $\kappa \in \{0,1\}^k$ or a special symbol $\top$ (indicating failure of the key establishment). If $\mathsf{2KEP}$ requires d rounds of communications, then GKEP takes $d+2$ rounds. Let $\mathcal{G} = \{U_1, \ldots, U_n\}$ be the set of n users wishing to establish a session key among themselves. Then GKEP proceeds as follows (throughout the protocol description, all indices are to be taken in a cycle, i.e., $U_{n+1} = U_1$, etc.):

Round $1 \sim d$**:** Each neighboring pair of users U_i and U_{i+1}, for $i = 1, \ldots, n$, generates a pairwise key $K_{i,i+1}$ by running the 2-party protocol $\mathsf{2KEP}$. (Accordingly, each U_i holds two pairwise keys $K_{i-1,i}$ and $K_{i,i+1}$ shared respectively with U_{i-1} and U_{i+1}.)

Round $d+1$**:**

Computation: Each U_i computes

$$X_i = K_{i-1,i} \oplus K_{i,i+1}$$

and chooses a random r_i to compute a commitment $C_i = C(i, X_i; r_i)$.

Transmission: Each U_i broadcasts $\textsc{Com}_i = \langle U_i \| C_i \rangle$.

Round $d+2$**:**

Transmission: Each U_i broadcasts $\textsc{Xor}_i = \langle U_i \| X_i \| r_i \rangle$.

Verification: Each U_i checks that $X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0$ and the correctness of the commitments. If any one of these checks fails, U_i terminates the protocol execution (without computing a session key).

Computation: Using $K_{i-1,i}$ and the XOR-values, each U_i computes

$$\begin{aligned} K_{i-2,i-1} &= X_{i-1} \oplus K_{i-1,i}, \\ K_{i-3,i-2} &= X_{i-2} \oplus K_{i-2,i-1}, \\ &\vdots \\ K_{i,i+1} &= X_{i+1} \oplus K_{i+1,i+2}. \end{aligned}$$

Then, U_i defines a master key

$$K = \langle K_{n,1} \| K_{1,2} \| \ldots \| K_{n-1,n} \| \mathcal{G} \rangle,$$

and computes the session key $SK_i = F_{H(K)}(v_1)$ and the session identifier $SID_i = F_{H(K)}(v_0)$.

Notice above that each X_i is never disclosed until the commitment C_j of every other X_j is received. This strategy, of course, prevents any potential attack where the value of some X_j is determined depending on the values of other X_i's. However, it is not sufficient enough to prevent our attack discussed in the next section.

For the instantiation of 2KEP, any particular choice that is, informally speaking, secure against an active adversary will do. If 2KEP is instantiated with the HMQV protocol of Krawczyk [3], the resulting GKEP is a 3-round group key exchange protocol where authentication is based on each user's private/public key pair.

3 Violating Implicit Key Authentication

Implicit key authentication is the fundamental security property that any given key exchange protocol is expected to achieve. Informally, this property means that no one outside the group can gain access to the session key. More formally:

Definition 1 (implicit key authentication) *Let $\mathcal{G}$ be a set of users who wish to share a session key by running a key exchange protocol P. Let sk_i be the session key computed by a user $U_i \in \mathcal{G}$ as a result of an execution of protocol P. We say that P achieves implicit key authentication if each $U_i \in \mathcal{G}$ is assured that no $U_k \notin \mathcal{G}$ can learn the key sk_i unless helped by a dishonest $U_j \in \mathcal{G}$.*

A key exchange protocol achieving implicit key authentication is said to be *authenticated*, and is a primitive of crucial importance in much of modern cryptography and network security.

Our main result is that the GKEP protocol fails to achieve authenticated key exchange. We prove this result by giving an active attack that violates implicit key authentication of GKEP. The adversary who mounts the attack is a legitimate user in the sense that she is able to set up normal protocol sessions with other users. Consider

a protocol session S to be conducted by the users of group $\mathcal{G} = \{U_1, \ldots, U_n\}$. Now suppose that U_{n-1} and U_n accept the invitation by the adversary U_{n+1} to participate in a new concurrent session S', thus forming the group $\mathcal{G}' = \{U_{n-1}, U_n, U_{n+1}\}$. Then, our attack is mounted (by U_{n+1}) against the session S, and leads to a serious consequence; at the end of the attack, the users in $\mathcal{G} \setminus \{U_n\}$ compute a common session key as per protocol specification and think that the session S is completed successfully, when, in fact, their session key is also known to the adversary U_{n+1}. Let U_i^S and $U_i^{S'}$ denote U_i's instances participating respectively in S and S'. The idea of our attack starts from the following observation:

> Since two neighboring users U_{n-1} and U_n participate in both sessions, they will try to establish two pairwise keys between themselves, specifically one between U_{n-1}^S and U_n^S and one between $U_{n-1}^{S'}$ and $U_n^{S'}$. But, notice that the adversary U_{n+1} can make U_{n-1}^S establish a pairwise key with $U_n^{S'}$ instead of with U_n^S and make $U_{n-1}^{S'}$ establish a pairwise key with U_n^S instead of with $U_n^{S'}$. A little thought will make it clear that no matter what 2-party key exchange protocol is selected as $\mathsf{2KEP}$, it is always possible for an active adversary to do so without being detected. All the adversary needs to do is to redirect the messages sent in two runs of $\mathsf{2KEP}(U_{n-1}, U_n)$ so that the messages from U_{n-1}^S (resp. U_n^S, $U_{n-1}^{S'}$, $U_n^{S'}$) are delivered to $U_n^{S'}$ (resp. $U_{n-1}^{S'}$, U_n^S, U_{n-1}^S).

This observation holds true as long as the two instances of U_{n-1} (and of U_n) use the same long-term secret(s) for authentication, which is the case for all of the three initial setups mentioned in the previous section.

We are now ready to describe the attack. It goes as follows:

1. As two sessions start, the adversary U_{n+1} interferes with the pairwise key establishments between U_{n-1} and U_n as in the observation above. Let $K_{n-1,n}^{SS'}$ (resp. $K_{n-1,n}^{S'S}$) be the pairwise key shared between U_{n-1}^S and $U_n^{S'}$ (resp. between $U_{n-1}^{S'}$ and U_n^S). All other pairwise keys of both sessions are generated without any interruption by U_{n+1}. Fig. 1 shows a snapshot of two sessions after all pairwise key establishments have been completed. Notice at this point that U_{n+1} holds two pairwise keys $K_{n,n+1}$ and $K_{n+1,n-1}$ shared respectively with $U_{n-1}^{S'}$ and $U_n^{S'}$.

2. For the rest of session S', U_{n+1} honestly interacts with $U_{n-1}^{S'}$ and $U_n^{S'}$. Let $X_{n-1}^{S'}$ (resp. $X_n^{S'}$) denote X_{n-1} (resp. X_n) sent by $U_{n-1}^{S'}$ (resp. $U_n^{S'}$). Then clearly, $X_{n-1}^{S'} = K_{n+1,n-1} \oplus K_{n-1,n}^{S'S}$ and $X_n^{S'} = K_{n-1,n}^{SS'} \oplus K_{n,n+1}$. Upon receiving $X_{n-1}^{S'}$ and $X_n^{S'}$, U_{n+1} computes the two pairwise keys

$$\begin{aligned} K_{n-1,n}^{S'S} &= X_{n-1}^{S'} \oplus K_{n+1,n-1}, \\ K_{n-1,n}^{SS'} &= X_n^{S'} \oplus K_{n,n+1}. \end{aligned}$$

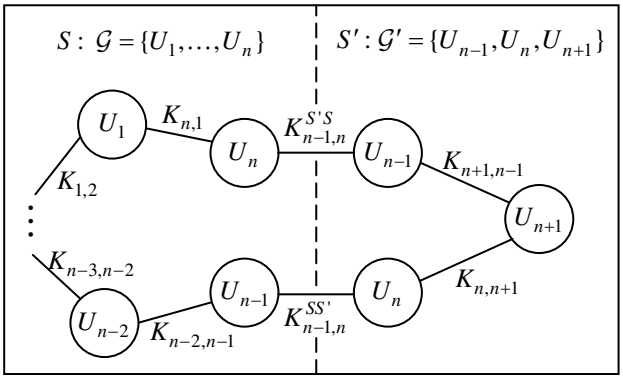

Figure 1: Two protocol sessions running concurrently in the presence of an active adversary U_{n+1}.

Of course, since $X_{n-1}^{S'} \oplus X_n^{S'} \oplus X_{n+1} \neq 0$, $U_{n-1}^{S'}$ and $U_n^{S'}$ will abort the protocol without computing a session key.

3. As the rest of session S proceeds, U_{n+1} intercepts U_n^S's first message $\textsc{Com}_n^S = \langle U_n \| C_n^S \rangle$. So, $\textsc{Com}_n^S$ is blocked from reaching its intended recipients. But, all other first messages (sent by $U_1, \ldots, U_{n-2}, U_{n-1}^S$) are transmitted without any interruption by U_{n+1}. As a result, U_n^S will receive all the $n-1$ first messages that are required to send its second message $\textsc{Xor}_n^S = \langle U_n \| X_n^S \| r_n^S \rangle$. As soon as $\textsc{Xor}_n^S$ is sent out, U_{n+1} intercepts it and computes

$$\begin{aligned} \overline{X}_n^S &= X_n^S \oplus K_{n-1,n}^{S'S} \oplus K_{n-1,n}^{SS'} \\ &= K_{n-1,n}^{SS'} \oplus K_{n,1}, \\ \overline{C}_n^S &= C(n, \overline{X}_n^S; r_n^S). \end{aligned}$$

Immediately after this computation is over, U_{n+1} sends $\overline{\textsc{Com}}_n^S = \langle U_n \| \overline{C}_n^S \rangle$ to $U_1, \ldots, U_{n-2}, U_{n-1}^S$ as the replacement for $\textsc{Com}_n^S$. Upon receiving $\overline{\textsc{Com}}_n^S$, the users $U_1, \ldots, U_{n-2}, U_{n-1}^S$ will send their second message. U_{n+1} eavesdrops these second messages while sending $\overline{\textsc{Xor}}_n^S = \langle U_n \| \overline{X}_n^S \| r_n^S \rangle$ as the replacement for $\textsc{Xor}_n^S$.

4. Since $X_1 \oplus \cdots \oplus X_{n-2} \oplus X_{n-1}^S \oplus X_n^S \neq 0$, U_n^S will abort the protocol without computing a session key. But since $X_1 \oplus \cdots \oplus X_{n-2} \oplus X_{n-1}^S \oplus \overline{X}_n^S = 0$ and $\overline{C}_n^S$ is the correct commitment of $\overline{X}_n^S$, the fake messages $\overline{\textsc{Com}}_n^S$ and $\overline{\textsc{Xor}}_n^S$ will pass the verification tests by $U_1, \ldots, U_{n-2}, U_{n-1}^S$. Hence, the users $U_1, \ldots, U_{n-2}, U_{n-1}^S$ will compute their session key as per the protocol specification. Notice that these

users should compute their $K_{n-1,n}$ as

$$\begin{aligned} K_{n-1,n} &= \overline{X}_n^S \oplus K_{n,1} \\ &= K_{n-1,n}^{SS'} \oplus K_{n,1} \oplus K_{n,1} \\ &= K_{n-1,n}^{SS'}. \end{aligned}$$

So, their master key is defined as

$$K = \langle K_{n,1} \| K_{1,2} \| \ldots \| K_{n-2,n-1} \| K_{n-1,n}^{SS'} \| \mathcal{G} \rangle.$$

Finally, their common session key SK is computed as $SK = F_{H(K)}(v_1)$. But this session key is also available to the adversary U_{n+1} who can compute all the pairwise keys required to derive the master key K.

At the end of the attack, each of $U_1, \ldots, U_{n-2}, U_{n-1}^S$ believes that they have established a secret key only with the rest of the group, while in fact the key has been shared also with the adversary U_{n+1}. This demonstrates that the GKEP protocol does not guarantee implicit key authentication when two protocol sessions are running concurrently with some joint participants. Implicit key authentication is the security property that is defined against an outsider adversary. Obviously, the adversary U_{n+1} is an outsider from the perspective of the users in $\mathcal{G}$.

Remark 1 *The attack above does not work in the setting where the complete set of protocol participants shares one common secret for authentication. To see this, suppose for example that the shared secret is a password. Then since each protocol session with different participants uses its respective password, the password shared between U_{n-1}^S and U_n^S is different (with overwhelming probability) from the one shared between $U_{n-1}^{S'}$ and $U_n^{S'}$. This means that the adversary U_{n+1} can no longer make U_{n-1}^S (resp. $U_{n-1}^{S'}$) establish a pairwise key with $U_n^{S'}$ (resp. U_n^S), and hence the attack fails.*

The GKEP protocol carries a claimed proof of its security in a formal model of communication and adversarial capabilities [1]. The proof model used for GKEP is a typical one and allows the adversary $\mathcal{A}$ to access all the standard oracles: Send, Execute, Reveal, Corrupt, and Test. Any key exchange protocol proven secure in such a model should certainly achieve implicit key authentication. But as we have seen, the GKEP protocol fails to achieve it. This implies that the security proof for GKEP is invalid. Indeed, the existence of our attack means, in the context of the proof model, that there exists an adversary $\mathcal{A}$ whose advantage in attacking protocol GKEP is 1, or in other words, there exists an adversary $\mathcal{A}$ who can distinguish, with probability 1, random keys from real session keys established by GKEP. The construction of such an $\mathcal{A}$ is rather straightforward from the attack above, and its brief description follows:

Corruption: First, $\mathcal{A}$ obtains all the long-term secrets of U_{n+1} by querying Corrupt(U_{n+1}).

Initiation: Next, $\mathcal{A}$ asks Send queries required to initiate two protocol sessions $S : \mathcal{G} = \{U_1, \ldots, U_n\}$ and $S' : \mathcal{G}' = \{U_{n-1}, U_n, U_{n+1}\}$. For example, a query of the form $\mathsf{Send}(U_{n-1}, *, \{U_n, U_{n+1}\})$ prompts an unused instance $*$ of U_{n-1} to initiate the protocol with U_n and U_{n+1}. But, no instance of U_{n+1} needs to be asked this form of Send query because $\mathcal{A}$ will simulate by itself the actions of U_{n+1}.

Run: Now, $\mathcal{A}$ runs the two sessions in the exact same way as U_{n+1} did in the above-described attack. Note that $\mathcal{A}$ can perfectly simulate U_{n+1}'s attack by asking Send queries and by using the disclosed long-term secrets (of U_{n+1}). Let U_i^S be U_i's instance participating in session S. Then, as in the attack above, the instances $U_1^S, \ldots, U_{n-1}^S$ will eventually accept a session key SK which can be also computed by $\mathcal{A}$.

Test: Clearly, all of the instances $U_1^S, \ldots, U_{n-1}^S$ are *fresh* (for the definition of freshness, see Section 2.2 of [1]); no Corrupt query has been asked for any of $U_1, \ldots, U_n$ and no Reveal query has ever been made for any instance. Thus, $\mathcal{A}$ may test (i.e., ask a Test query against) any of the $n-1$ instances. Since $\mathcal{A}$ knows the value of SK, the probability that $\mathcal{A}$ guesses correctly the bit b used by the Test oracle is 1 and so is the advantage of $\mathcal{A}$ in attacking GKEP.

4 Fixing the Protocol

Given the attack, it is quite apparent how to repair the GKEP protocol. Before each user U_i sends X_i, they first have to make sure that their pairwise keys have been established as originally intended. A simple way to ensure this is to let the users carry out the following procedure for pairwise key confirmation.

Pairwise key confirmation: Each U_i computes $\overleftarrow{\sigma}_i = F_{H(U_i \| K_{i-1,i} \| \mathcal{G})}(v_0)$ and $\overrightarrow{\sigma}_i = F_{H(U_i \| K_{i,i+1} \| \mathcal{G})}(v_0)$, and sends $\overleftarrow{\textsc{Auth}}_i = \langle U_i \| \overleftarrow{\sigma}_i \rangle$ and $\overrightarrow{\textsc{Auth}}_i = \langle U_i \| \overrightarrow{\sigma}_i \rangle$ respectively to U_{i-1} and U_{i+1}. On receiving $\overrightarrow{\textsc{Auth}}_{i-1}$ and $\overleftarrow{\textsc{Auth}}_{i+1}$, U_i verifies the correctness of both $\overrightarrow{\sigma}_{i-1}$ and $\overleftarrow{\sigma}_{i+1}$ in the straightforward way. If either one of the verifications fails, U_i aborts the protocol.

Adding the confirmation procedure above into the protocol does not necessarily cause any additional round of communication. Since the pairwise key confirmation should be done before sending the XOR-values and after establishing the pairwise keys, we can simply integrate the confirmation procedure into the $(d+1)^{\text{th}}$ round of the protocol as follows:

Round $d+1$ (revision):

Computation: Each U_i computes

$$X_i = K_{i-1,i} \oplus K_{i,i+1}$$

and chooses a random r_i to compute a commitment $C_i = C(i, X_i; r_i)$. In addition, U_i computes $\overleftarrow{\sigma}_i = F_{H(U_i \| K_{i-1,i} \| \mathcal{G})}(v_0)$ and $\overrightarrow{\sigma}_i = F_{H(U_i \| K_{i,i+1} \| \mathcal{G})}(v_0)$.

Transmission: Each U_i broadcasts $\textsc{Com}_i = \langle U_i \| C_i \rangle$, and sends $\overleftarrow{\textsc{Auth}}_i = \langle U_i \| \overleftarrow{\sigma}_i \rangle$ and $\overrightarrow{\textsc{Auth}}_i = \langle U_i \| \overrightarrow{\sigma}_i \rangle$ respectively to U_{i-1} and U_{i+1}.

Verification: Upon receiving $\overrightarrow{\textsc{Auth}}_{i-1}$ and $\overleftarrow{\textsc{Auth}}_{i+1}$, U_i checks the correctness of both $\overrightarrow{\sigma}_{i-1}$ and $\overleftarrow{\sigma}_{i+1}$ in the straightforward way. If either one of the checks fails, U_i aborts the protocol.

The other rounds of the protocol remain unchanged.

Our modification effectively prevents the attack. Notice in our pairwise key confirmation that the authenticators $\overleftarrow{\sigma}_i$ and $\overrightarrow{\sigma}_i$ are computed by taking as input the set of the identities of all group members. This way of computing authenticators plays a critical role in preventing the attack. Suppose that the adversary U_{n+1} interferes with the pairwise key establishments as described in the attack. Then, no authenticators exchanged between the instances of U_{n-1} and U_n can pass the verification step of the revised $(d+1)^{\text{th}}$ round. As soon as the verifications fail, the instances of U_{n-1} and U_n will abort the protocol without revealing their XOR-value of two pairwise keys. Hence, the attack is not valid against the improved protocol.

Another way to prevent the attack is to modify the underlying 2-party protocol 2KEP so that the identities of all group members are included in computing every authentication message of 2KEP. But then the construction would be no longer generic.

References

[1] M. Abdalla, J.-M. Bohli, M. Vasco, R. Steinwandt, (Password) authenticated key establishment: from 2-party to group, in: Proceedings of the 4th Theory of Cryptography Conference (TCC'07), Lecture Notes in Computer Science, vol. 4392, 2007, pp. 499–514.

[2] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (6) (1976) 644–654.

[3] H. Krawczyk, HMQV: a high-performance secure Diffie-Hellman protocol, in: Advances in Cryptology — CRYPTO '05, Lecture Notes in Computer Science, vol. 3621, 2005, pp. 546–566.

[4] J. Nam, J. Paik, U. Kim, D. Won, Security enhancement to a password-authenticated group key exchange protocol for mobile ad-hoc networks, IEEE Communications Letters, 12 (2) (2008) 127–129.

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

[6] K. Shim, S. Woo, Cryptanalysis of tripartite and multi-party authenticated key agreement protocols, Information Sciences 177 (4) (2007) 1143–1151.