Journal of Computing and Security

January 2018, Volume 5, Number 1 (pp. 3–12)
http://www.jcomsec.org

An Improvement Over Lee et al.'s Key Agreement Protocol

Hossein Oraei [a], Mohsen Pourpouneh [b], Rasoul Ramezanian [c,*],

[a] *Department of Mathematical Sciences, Iran University of Science & Technology, Tehran, Iran.*
[b] *Department of Mathematical Sciences, Sharif University of Technology, Tehran, Iran.*
[c] *Department of Mathematical Sciences, Ferdowsi University of Mashhad, Mashhad, Iran.*

ARTICLE INFO.

Article history:
Received: 16 July 2017
Revised: 01 October 2018
Accepted: 03 October 2018
Published Online: 26 December 2018

Keywords:
Backward Secrecy, Forward Secrecy, Group Key Agreement, Scyther.

ABSTRACT

In 2004, Hwang et al. proposed a group key exchange protocol for sharing a secure key in a group. Their protocol is an extension from the two party key exchange protocol to the group one. Recently, Jung-San Lee et al. noted that Hwang et al. group key exchange protocol has two security weaknesses. First, the forward secrecy is not confirmed in case that a new member joins the group and second, if a group member leaves the group, the backward secrecy is compromised. They proposed an improvement over this key exchange protocol in order to provide both forward and backward secrecy among group members. In this paper, we propose another improvement over Lee et al. key exchange, and we show that our key exchange protocol not only preservers both forward and backward secrecy, but also it is more efficient than their protocol when a member leaves the group. Finally, we give a formal analysis for the correctness of the proposed protocol via Scyther model checking tool.

1 Introduction

Key exchange protocols are one of the most important issues in cryptography. There are different protocols that are employed in order to generate session keys among a group of agents. These protocols usually fall in to two main categories: Key transport protocols and Key agreement protocols. In a key transport protocol a trusted party (key generation center) generates a session key and then securely transfers it to every group member. In contrast, in a key agreement protocol the session key is a function of all users' inputs. One of the well-known protocols of this category is the "Diffie-Hellman" key exchange protocol, which is proposed in 1976 [1].

* Corresponding author.

Email addresses: `hossein_oraei@mathdep.iust.ac.ir` (H. Oraei), `mohsen.pourpoune1@student.sharif.ir` (M. Pourpouneh), `rramezanian@um.ac.ir` (R. Ramezanian)

https://dx.doi.org/10.22108/jcs.2018.106883.0
ISSN: 2322-4460 © 2018 JComSec. All rights reserved.

Message confidentiality is a property which ensures the sender the data can only be read by an authorized and intended receiver. Message confidentiality is mainly achieved by two components namely Forward Secrecy and Backward Secrecy. The main idea of forward is that if an adversary knows a set of long-term keys then he should not be able to find any subsequent keys. Also, backward secrecy requires that if an adversary knows a set of group keys then he should not be able to find any earlier group key. More specifically, in our context

- Forward Secrecy, is a mechanism to ensure that whenever the user leaves the group he will not have any access to the future key [2].
- Backward Secrecy, ensures that whenever a new user joins the group, he will not get any access to the previous details [3].

Most distributed group key management protocols are natural generalization of the Diffie-Hellman key agreement protocol, as an example [4–7] follow this approach. In 1996, Steiner et al. proposed the group Diffie-Hellman key exchange [7] protocol. Later in 2001, it has been enhanced with authentication services and has proved to be secure in [8]. In 2006, Bohli [9] developed a framework for robust group key agreement that provides security against malicious insiders and active adversaries in an unauthenticated point-to-point network. Then, in 2001, Bresson et al. [10] constructed a generic authenticated group Diffie-Hellman key exchange and showed that it is provably secure. Also, in 2003, Katz and Yung [11] proposed the first constant-round and fully scalable group Diffie-Hellman protocol which is provably secure in the standard model. The main feature of the group Diffie-Hellman key exchange is to establish a secret group key among all group members without relying on a trusted server.

Mayer and Yung have proposed a compiler to convert the two-party key exchange protocol to the group one [12]. The group key exchange architecture proposed by Mayer and Yung is a key transfer protocol. Key transfer approach has two main weaknesses. First, it is un-scalable for large groups. Second, the failure of the centralized controller will lead to the failure of the whole group communication.

Hwang et al. [13] proposed a distributed group key exchange mechanism using the compiler suggested by Katz and Yung [11], in which each group member has to take the responsibility for key generation and maintaining the security of the whole group. In 2013 Jung-San Lee et. al. [14] found that Hwang et al.'s group key exchange protocol has two security weaknesses. First, the forward secrecy is not confirmed when a new member joins the communication group. That is, the new member can compromise advanced group secret keys to retrieve the previous messages shared between previous group members. Second, if a group member is expelled or leaves the group, the backward secrecy is not preserved. In other words, the expelled or leaving member can compromise oncoming group secret keys to learn the shared messages in the future. They proposed a new secure conference key distribution mechanism (SCKDM) in order to improve the security weaknesses of Hwang et al.'s group key exchange protocol.

In this paper we improve the efficiency of Jung-San Lee et. al's protocol when a member leaves the group, in the way that the remaining members do not have to re-run the entire protocol after a member leaves the group.

The rest of this paper is organized as follows: In Section 2, we recall Jung-San Lee et al.'s key agreement protocol. Then we present our improved protocol in Section 3. The security analysis are presented in Section 4. Finally, the conclusion is drawn in Section 5.

Table 1. The Notations.

Notation	Description
n	number of users
U_i	identity of the user i, where $U_1, ..., U_n$ are in a predefined order
p	a large prime (usually at least 1024 bits)
q	a prime (typically of 160 bits) with $q\|p-1$
$\mathbb{Z}_p$	field of integers (under addition and multiplication) modulo P
$\mathbb{Z}_p^*$	multiplicative group of non-zero integers modulo p
$K(i, i+1)$	secret key shared between U_i and U_{i+1}, where $K(i, i+1) \in \mathbb{Z}_p^*$
$K(n, 1)$	secret key shared between U_n and U_1, where $K(n, 1) \in \mathbb{Z}_p^*$
$K(i, i+1)^{-1}$	multiplicative inverse of $K(i, i+1)$ in the multiplicative group $\mathbb{Z}_p^*$
$K(n, 1)^{-1}$	multiplicative inverse of $K(n, 1)$ in the multiplicative group $\mathbb{Z}_p^*$
$H(.)$	a public one-way hash function
$\oplus$	exclusive-or operation
r_i	random integer chosen by user i
r'_i	arbitrary integer chosen by user i
s'	a public shared value calculated by the users in the initiation phase
sk	the derived session key

2 A Review of Jung-San Lee et al.'s (LCW) Key Agreement Protocol

In this section, we review Jung-San Lee et al.'s (LCW) key agreement protocol. Afterwards, we improve the member leave operation of this protocol.

Assumed that a secure Diffie-Hellman two-party key exchange protocol is available, the Jung-San Lee et al.'s key agreement protocol has three main operations: 1- the group key exchange operation, 2- the member join operation and 3- the member leave operation. We use the notations in Table 1 throughout this paper:

2.1 The Group Key Exchange Operation

This operation, which is the key establishment phase, has four steps.

1) Each group member U_i performs the secure Diffie-Hellman two-party key exchange protocol with his neighbors U_{i-1} and U_{i+1}, and then negotiates the secret keys $K(i-1, i)$ and $K(i, i+1)$, respectively.
2) Each U_i computes $Z_i = K(i-1, i) \oplus K(i, i+1)$ and then broadcasts the computation result to

all group members, where $i = 2, 3, \ldots, n-1$. Note that, Z_1 and Z_n are computed as $Z_1 = K(n,1) \oplus K(1,2)$ and $Z_n = K(n-1,n) \oplus K(n,1)$, respectively.

3) After receiving all Z_j's, each U_j can get other secret keys $K(j, j+1)$'s by means of the following inference, where $j = 1, 2, ..., n$.

$$\begin{aligned} &K(j, j+1) \\ &K(j+1, j+2) = Z_{j+1} \oplus K(j, j+1) \\ &K(j+2, j+3) = Z_{j+2} \oplus K(j+1, j+2) \\ &\quad\vdots \qquad \vdots \qquad \vdots \\ &K(j-1, j) = Z_{j-1} \oplus K(j-1, j) \end{aligned}$$

4) After collecting all secret keys, U_i can compute the group session key as follows,

$$sk = H(K(1,2), K(2,3), \ldots, K(n-1,n), K(n,1)).$$

2.2 The Member Join Operation

This operation makes the group key exchange mechanism be able to confirm the forward secrecy. While a new user U_{n+1} joins into the communication group, the secret keys $K(n, n+1)$ and $K(n+1, 1)$ are generated for U_{n+1}. Besides, Z_n is updated as

$$Z_n = K(n-1, n) \oplus K(n, n+1),$$

and Z_{n+1} is constructed as

$$Z_{n+1} = K(n, n+1) \oplus K(n+1, 1).$$

Finally, the current group session key is computed as

$$sk = H(K(1,2), K(2,3), .., K(n, n+1), K(n+1, 1)).$$

Now each group member can obtain other participants' secret keys by performing Step 3 of the group key exchange operation.

2.3 The Member Leave Operation

This operation makes the group key exchange mechanism be able to preserve the backward secrecy. While a member U_d leaves the communication group, U_{d-1} and U_{d+1} have to negotiated a new secret key $K_{new}(d-1, d+1)$ by performing a secure Diffie-Hellman two party key exchange protocol defined in previous section. At the same time, other group members have to perform Step 1 of the group key exchange operation. Then, the new group session key is computed as

$$sk = H(K_{new}(1,2), K_{new}(2,3), \ldots, K_{new}(d-2, d-1), K_{new}(d-1, d+1), K_{new}(d+1, d+2), \ldots, K_{new}(n-1, n), K_{new}(n, 1)),$$

where $K_{new}(j, j+1)$'s and $K_{new}(n,1)$ are the new generated secret keys for $j = 1, 2, \ldots, n-1$ and $j \neq d-1, d$.

Member leave operation of Jung-San Lee et al.'s (LCW) key agreement protocol is not efficient in the sense that when a member leaves the group, each remaining member, say U_i have to run the secure Diffie-Hellman two-party protocol with his neighbors again to share new secret keys. We improve the member leave operation in the way that it is not needed that all the remaining member have to run the secure Diffie-Hellman two-party protocol.

3 Our Proposed Scheme

In this section, we describe the model of our group key agreement protocol which is an improved protocol based on Hwang et al.'s scheme. Then, we present the security notes for it. In our improved protocol, we keep the notations introduced in the last section. Our group key agreement protocol has three main operations: the session key agreement operation, the member join operation and the member leave operation.

3.1 The Session Key Agreement Operation

Here, The session key agreement operation is presented to makes all group members be able to establish a common session key. This operation consists of two phases: the initiation phase and the session key agreement phase. The details are included in the following.

3.1.1 The Initiation Phase

This phase makes all group members be able to compute the public shared value s' and has three steps.

1) Each group member U_i performs the secure Diffie-Hellman two-party key exchange protocol with his neighbors U_{i-1} and U_{i+1}, and then negotiates the secret keys $K(i-1, i)$ and $K(i, i+1)$, respectively. Here, the algebraic group G in the Diffie-Hellman key agreement protocol is G_0, where G_0 is a subgroup of $\mathbb{Z}_p^*$ of prime order q [15].
2) Each U_i computes $Z'_i = K(i-1, i) \oplus K(i, i+1) \oplus r_i$ where $(2 \leq i \leq n-1)$. But U_1 and U_n compute $Z'_1 = r'_1 \oplus K(1,2) \oplus r_1$ and $Z'_n = K(n-1, n) \oplus K(n,1) \oplus r_n$, respectively. Then each group member U_i broadcasts the computation result to all group members $(1 \leq i \leq n)$:

$$\begin{aligned}
Z'_1 &= r'_1 \oplus K(1,2) \oplus r_1 \\
Z'_2 &= K(1,2) \oplus K(2,3) \oplus r_2 \\
&\vdots \quad \vdots \quad \vdots \\
Z'_i &= K(i-1,i) \oplus K(i,i+1) \oplus r_i \\
&\vdots \quad \vdots \quad \vdots \\
Z'_{n-1} &= K(n-2,n-1) \oplus K(n-1,n) \oplus r_{n-1} \\
Z'_n &= K(n-1,n) \oplus K(n,1) \oplus r_n
\end{aligned}$$

Here, r'_1 is chosen by U_1 such that $A = r'_1 \oplus K(n,1) \in G_0$ (Note that it is possible. Sinse, r'_1 is an arbitrary integer).

3) After receiving all Z'_i's, each U_i computes $s' = Z'_1 \oplus Z'_2 \oplus ... \oplus Z'_n$ and obtains

$$s' = A \oplus B$$

, where $A = r'_1 \oplus K(n,1)$ and $B = r_1 \oplus r_2 \oplus ... \oplus r_n$.

3.1.2 The Session Key Agreement Phase

The session key agreement phase makes all group members be able to compute the session key sk.

In this phase, first the group member U_1 computes $Z_1 = K(1,2) \oplus (A \times K(1,2))$ and broadcasts the computation result, where $\times$ is the multiplication operation of $\mathbb{Z}_p$ (note that U_1 can compute $A = r'_1 \oplus K(n,1)$). Then, U_2 computes $A = (Z_1 \oplus K(1,2)) \times K(1,2)^{-1}$ and broadcasts $Z_2 = K(2,3) \oplus (A \times K(2,3))$. Next, U_3 computes $A = (Z_2 \oplus K(2,3)) \times K(2,3)^{-1}$ and broadcasts $Z_3 = K(3,4) \oplus (A \times K(3,4))$. Actually, each group member U_i computes $A = (Z_{i-1} \oplus K(i-1,i)) \times K(i-1,i)^{-1}$ and broadcasts $Z_i = K(i,i+1) \oplus (A \times K(i,i+1))$, respectively where $i \in \{2,3,...,n-1\}$. Finally, U_n computes $A = (Z_{n-1} \oplus K(n-1,n)) \times K(n-1,n)^{-1}$:

$$\begin{aligned}
Z_1 &= K(1,2) \oplus (A \times K(1,2)) \\
Z_2 &= K(2,3) \oplus (A \times K(2,3)) \\
&\vdots \quad \vdots \quad \vdots \\
Z_{n-1} &= K(n-1,n) \oplus (A \times K(n-1,n))
\end{aligned}$$

Now,each group member U_i can obtain $B = s' \oplus A$. The group session key is

$$sk = H(A,B).$$

3.2 The Member Join Operation

This operation makes the group key exchange mechanism be able to confirm the forward secrecy. While a new user U_{n+1} joins into the communication group, the secret keys $K(n,n+1)$ and $K(n+1,1)$ are generated for U_{n+1}. Besides, Z'_n is updated as

$$Z'_n = K(n-1,n) \oplus K(n,n+1) \oplus r_n$$

and Z'_{n+1} is constructed as

$$Z'_{n+1} = K(n,n+1) \oplus K(n+1,1) \oplus r_{n+1}.$$

Now, all group members have to perform Step 3 of the initiation phase of the session key agreement operation and the session key agreement phase.

Finally, the new group session key is computed as

$$sk = H(A,B),$$

where $A = r'_1 \oplus K(n+1,1)$ and $B = r_1 \oplus r_2 \oplus ... \oplus r_{n+1}$.

3.3 The Member Leave Operation

This operation makes the group key exchange mechanism be able to preserve the backward secrecy. Here, we can consider five cases.

(1) If U_1 leaves the communication group, the secret key $K(n,2)$ is generated. Besides, Z'_2 and Z'_n are updated as

$$Z'_2 = r'_2 \oplus K(2,3) \oplus r_2$$

and

$$Z'_n = K(n-1,n) \oplus K(n,2) \oplus r_n,$$

where r'_2 is chosen by U_2 such that $A = r'_2 \oplus K(n,2) \in G_0$. Now, all group members have to perform Step 3 of the initiation phase of the session key agreement operation and the session key agreement phase.

The new group session key is computed as

$$sk = H(A,B),$$

where $A = r'_2 \oplus K(n,2)$ and $B = r_2 \oplus r_3 \oplus ... \oplus r_n$.

(2) If U_2 leaves the communication group, the secret key $K(1,3)$ is generated. Besides, Z'_1 and Z'_3 are updated as

$$Z'_1 = r''_1 \oplus K(1,3) \oplus r_1$$

and

$$Z'_3 = K(1,3) \oplus K(3,4) \oplus r_3,$$

where r''_1 is a new arbitary intger chosen by U_1 such that $A = r''_1 \oplus K(n,1) \in G_0$.

Now, all group members have to perform Step 3 of the initiation phase of the session key agreement operation and the session key agreement phase.

The new group session key is computed as

$$sk = H(A,B),$$

where $A = r''_1 \oplus K(n,1)$ and $B = r_1 \oplus r_3 \oplus r_4 \oplus ... \oplus r_n$.

(3) If U_{n-1} leaves the communication group, the secret key $K(n-2,n)$ is generated. Besides, Z'_1, Z'_{n-2} and Z'_n are updated as

$Z'_1 = r''_1 \oplus K(1,2) \oplus r_1,$
$Z'_{n-2} = K(n-3,n-2) \oplus K(n-2,n) \oplus r_{n-2}$
and
$Z'_n = K(n-2,n) \oplus K(n,1) \oplus r_n,$

where r''_1 is a new arbitary intger chosen by U_1 such that $A = r''_1 \oplus K(n,1) \in G_0$. Now, all group members have to perform Step 3 of the initiation phase of the session key agreement operation and the session key agreement phase.

The new group session key is computed as

$$sk = H(A,B),$$

where $A = r''_1 \oplus K(n,1)$ and $B = r_1 \oplus r_2 \oplus ... \oplus r_{n-1} \oplus r_n$.

(4) If U_n leaves the communication group, the secret key $K(n-1,1)$ is generated. Besides, Z'_1 and Z'_{n-1} are updated as

$Z_1 = r''_1 \oplus K(1,2) \oplus r_1,$
$Z'_{n-1} = K(n-2,n-1) \oplus K(n-1,1) \oplus r_{n-1}.$

where r''_1 is a new arbitary intger chosen by U_1 such that $A = r''_1 \oplus K(n-1,1) \in G_0$. Now, all group members have to perform Step 3 of the initiation phase of the session key agreement operation and the session key agreement phase.

The new group session key is computed as

$$sk = H(A,B),$$

where $A = r''_1 \oplus K(n-1,1)$ and $B = r_1 \oplus r_2 \oplus ... \oplus r_{n-1}$.

(5) And if U_i $(i \in \{3,4,...,n-2\})$ leaves the communication group, the secret key $K(i-1,i+1)$ is generated. Besides, Z'_1, Z'_{i-1} and Z'_{i+1} are updated as

$Z'_1 = r''_1 \oplus K(1,2) \oplus r_1,$
$Z'_{i-1} = K(i-2,i-1) \oplus K(i-1,i+1) \oplus r_{i-1}$
and
$Z'_{i+1} = K(i-1,i+1) \oplus K(i+1,i+2) \oplus r_{i+1},$

where r''_1 is a new arbitary intger chosen by U_1 such that $A = r''_1 \oplus K(n,1) \in G_0$. Now, all group members have to perform Step 3 of the initiation phase of the session key agreement operation and the session key agreement phase.

Finally, the new group session key is computed as

$$sk = H(A,B),$$

where $A = r''_1 \oplus K(n,1)$ and $B = r_1 \oplus r_2 \oplus ... \oplus r_{i-1} \oplus r_{i+1} \oplus r_{i+2} \oplus ... \oplus r_n$.

In the rest of this section, we refer to two important notes.

- In this scheme all r'_i's and r''_i's are chosen such that $A \in G_0$.
- We know that in a field, there is no divisor of zero. Thus, $A \times K(i.i+1) \neq 0$ (Note that $A, K(i,i+1) \in G_0$. So, $A, K(i,i+1) \neq 0$).

Hence, there is no problem in the key establishment.

Table 2. The Number of Key Exchanges

Operation	Lee et al.[14]	Proposed Protocol
Group Key Exchange	nT_d	nT_d
Member Join	$2T_d$	$2T_d$
Member Leave	$(n-1)T_d$	T_d

4 Analysis

4.1 Computational Analysis

In the Jung-San Lee et al.'s protocol, when a member leaves the group, the other members, that are still in the group, have to perform the the entire group key exchange protocol. In particular, they have to perform the secure Diffie-Hellman two-party key exchange protocol with their neighbors (step 1 of the group key exchange operation). So the member leave operation of the Jung-San Lee et al.'s protocol is not efficient. While in our proposed scheme, when a member leaves the group, other group members do not perform step 1 of the initiation phase of the session key agreement operation. In fact, our scheme offers efficiency when a member leaves the group, while Jung-San Lee et al.'s protocol does not provide this property.

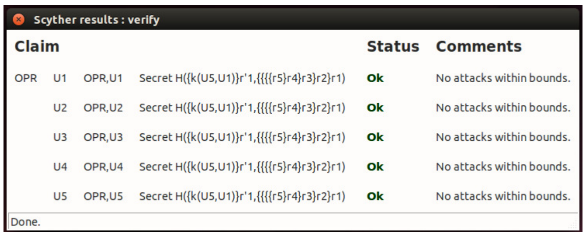
The results of more detailed comparison are shown in Table 2 and Table 3. In Table 2 we compare the protocols based on the number of Diffie Hellman key exchange that are necessary to perform the protocols, and in Table 3 we compare the number of arithmetic operations that are required in each protocol. In these tables T_d denoted the computational cost for executing the Diffie Hellman protocol, T_e denotes the computational cost of XoR operation. Also, T_m denotes the computational cost of multiplication in the field $\mathbb{Z}_p$ and T_h denotes the computational cost of the hash function.

4.2 Security Analysis

In this section, we model and analyze our proposed protocol by the Scyther verification tool [16]. We first show that the shared session key of the proposed scheme is secret. Then we provide an authenticated version of our proposed scheme which has the Non-injective synchronisation property [17].

Table 3. Computational Complexity for Necessary Operation for Each User U_i.

Operation	Lee et al.[14]	Proposed Protocol
Group Key Exchange	$(n-1)T_e + T_h$	$(n+3)T_e + 2T_m + T_h$
Member Join	$3T_e + T_h$	$10T_e + 2T_m + T_h$
Member Leave	$(n-2)T_e + T_h$	$8T_e + 2T_m + T_h$



```
Scyther results : verify
Claim                                                    Status  Comments
OPR  U1  OPR,U1  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
     U2  OPR,U2  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
     U3  OPR,U3  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
     U4  OPR,U4  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
     U5  OPR,U5  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
Done.
```

Figure 1. Secrecy Claim. Setup: Maximum Number of runs = 1, Matching type = typed matching, Search pruning = Find All Attacks.

4.2.1 Analyzing Security of Shared Session Key

We prove the secrecy of the shared session key by using Scyther model checking tool. Because of some limitations of Scyther, we have to use an abstraction of our protocol. The most important issue in this modeling, is that Scyther does not support "message broadcasting ". In order to simulate broadcasting in Scyther, when a group member U_i wants to broadcasts a message to all members, instead of it we let him to send his message to all members $U_{j\in\{1,2,\ldots,i-1,i+1,\ldots,n\}}$. The Scyther code of of our protocol is given in Appendix A. The results of running Scyther are shown in Figure 1 which shows that the shared session key of the proposed scheme is secret.

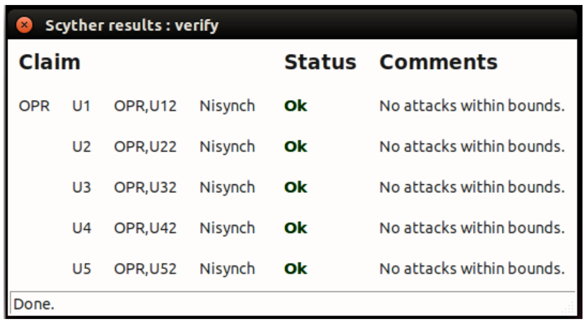
4.2.2 Analyzing Non-Injective Synchronisation Property

The model of our group key agreement protocol does not provide Non-injective synchronisation property. In order to provide Non-injective synchronisation we use a secure HMAC in structre of messages. The Scyther code of abstraction of modified protocol is given in Appendix B. The results of running Scyther is shown in Figure 2 and Figure 3,

thereby proving that the modified protocol provides Non-injective synchronisation property.

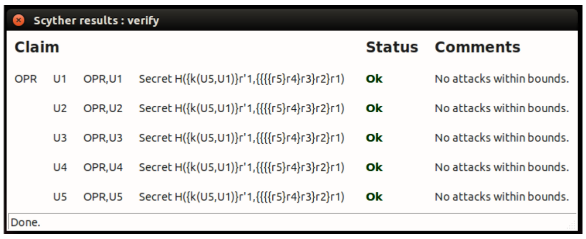
4.2.3 Analyzing The Backward Secrecy Property

In this subsection, we show that our proposed scheme can confirm the backward secrecy by means of the member join operation. While a new user U_{n+1} joins into the communication group, the secret keys



```
Scyther results : verify
Claim                        Status  Comments
OPR  U1  OPR,U12  Nisynch    Ok      No attacks within bounds.
     U2  OPR,U22  Nisynch    Ok      No attacks within bounds.
     U3  OPR,U32  Nisynch    Ok      No attacks within bounds.
     U4  OPR,U42  Nisynch    Ok      No attacks within bounds.
     U5  OPR,U52  Nisynch    Ok      No attacks within bounds.
Done.
```

Figure 2. Ni-Synch Claim. Setup: Maximum Number of runs = 1, Matching type = typed matching, Search pruning = Find All Attacks.



```
Scyther results : verify
Claim                                                    Status  Comments
OPR  U1  OPR,U1  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
     U2  OPR,U2  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
     U3  OPR,U3  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
     U4  OPR,U4  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
     U5  OPR,U5  Secret H({k(U5,U1)}r'1,{{{{r5}r4}r3}r2}r1)  Ok  No attacks within bounds.
Done.
```

Figure 3. Secrecy Claim. Setup: Maximum Number of runs = 1, Matching type = typed matching, Search pruning = Find All Attacks.

$K(n,n+1)$ and $K(n+1,1)$ are generated for U_{n+1}. Besides, Z'_n is updated as

$$Z'_n = K(n-1,n) \oplus K(n,n+1) \oplus r_n$$

instead of

$$Z'_n = K(n-1,n) \oplus K(n,1) \oplus r_n$$

,

and Z'_{n+1} is constructed as

$$Z'_{n+1} = K(n,n+1) \oplus K(n+1,1) \oplus r_{n+1}.$$

Now, all group members have to perform Step 3 of the initiation phase of the session key agreement operation and the session key agreement phase.

Finally, the new group session key is computed as

$$sk = H(A,B),$$

where $A = r'_1 \oplus K(n+1,1)$ and $B = r_1 \oplus r_2 \oplus \ldots \oplus r_{n+1}$. It is worth noting that U_{n+1} can not recover the past group session keys shared among U_1 to U_n. It is due to that U_{n+1} can not obtain $K(n,1)$ to compute those past session keys by means of the following computation.

$$sk = H(A, B),$$

where $A = r'_1 \oplus K(n, 1)$. As a result, the backward secrecy is preserved in our proposed scheme.

4.2.4 Analyzing The Forward Secrecy Property

Here, we show that our scheme can confirm the forward secrecy. In case that a member U_d leaves the communication group ($d \in \{1, 2, ..., n\}$), as mentioned in subsection 3.3, the value of A is updated. Then, the new group session key is computed as

$$sk = H(A, B).$$

Without the knowledge of new updated A, U_d can not compromise the current group session key sk. Thereby, the forward secrecy is preserved in our proposed scheme.

5 Conclusion

In this paper, we have presented an improved protocol which can preserve the forward secrecy and the backward secrecy. The correctness of the proposed scheme is formally analyzed by Scyther. Furthermore, we have shown that the improved protocol offers efficiency when a member leaves the group, while Jung-San Lee et al.'s protocol does not provide this property.

References

[1] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644 – 654, 1976. ISSN 1557-9654. doi:10.1109/TIT.1976.1055638.

[2] Adrian Perrig. Efficient collaborative key management protocols for secure autonomous group communication. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99)*, pages 192–202, 1999. ISBN 1-58113-148-8. doi:10.1.1.42.4440.

[3] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769 – 780, 2000. ISSN 1558-2183. doi:10.1109/71.877936.

[4] I. Ingemarsson, D. Tang, and C. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5):714–720, 1982. ISSN 1557-9654. doi:10.1109/TIT.1982.1056542.

[5] D. G. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In *Proceedings on Advances in cryptology*, pages 520–528. Springer-Verlag New York, Inc., 1990. ISBN 0-387-97196-3.

[6] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology EUROCRYPT'94*, pages 275–286. Springer, Berlin, Heidelberg, 1995. ISBN 978-3-540-44717-7. doi:10.1007/BFb0053443.

[7] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 31 – 37. ACM, 1996. ISBN 0-89791-829-0. doi:10.1145/238168.238182.

[8] E. Bresson, O. Chevassut, D. Pointcheval, and J. Jacques Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 255–264. ACM, 2001. ISBN 1-58113-385-5. doi:10.1145/501983.502018.

[9] Jens-Matthias Bohli. A framework for robust group key agreement. In *Computational science and its applications-ICCSA 2006*, pages 355–364. Springer, Berlin, Heidelberg, 2006. ISBN 978-3-540-34076-8. doi:10.1007/11751595_39.

[10] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 255–264. ACM, 2001. ISBN 1-58113-385-5. doi:10.1145/501983.502018.

[11] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *Advances in cryptology-CRYPTO 2003*, pages 110–125. Springer, Berlin, Heidelberg, 2003. ISBN 978-3-540-45146-4. doi:10.1007/978-3-540-45146-4_7.

[12] A. Mayer and M. Yung. Secure protocol transformation via "expansion": from two-party to groups. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 83 – 92. ACM, 1999. ISBN 1-58113-148-8. doi:10.1145/319709.319721.

[13] J.Y. Hwang, S.M. Lee, and D.H. Lee. Scalable key exchange transformation: from two-party to group. *Electronics Letters*, 40(12):728 – 729, 2004. ISSN 0013-5194. doi:10.1049/el:20040449.

[14] J. S Lee, C. C Chang, and K. J Wei. Provably secure conference key distribution mechanism preserving the forward and backward secrecy. *International Journal of Network Security*, 15(5):405 – 410, 2013. doi:10.6633/IJNS.201309.15(5).11.

[15] C. Boyd and A. Mathuria. *Protocols for authentication and key establishment*. Springer, 2013.

[16] Cas J. F. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification*, pages 414–418. Springer, Berlin, Heidelberg, 2008. ISBN 978-3-

540-70545-1. doi:10.1007/978-3-540-70545-1_38.
[17] C. Cremers, , and S. Mauw. *Operational semantics and verification of security protocols*. Springer, 2012.

Appendix A

The Scyther verification code for secrecy claim:

```
const Star: Function;
hashfunction H;
usertype XOR;
protocol OPR(U1,U2,U3,U4,U5){
role U1 {
    fresh r1 : XOR;
    fresh r'1 : XOR;
    var T1-2, T1-3, T1-4, T1-5 : Ticket;
    var r2, r3, r4, r5 : XOR;
    send_1(U1,U2, {{k(U1,U2)}r'1}r1 );
    send_2(U1,U3, {{k(U1,U2)}r'1}r1 );
    send_3(U1,U4, {{k(U1,U2)}r'1}r1 );
    send_4(U1,U5, {{k(U1,U2)}r'1}r1 );
    recv_5(U2,U1, {T1-2}r2);
    recv_9(U3,U1, {T1-3}r3);
    recv_13(U4,U1, {T1-4}r4 );
   recv_17(U5,U1, {T1-5}r5);
        send_21(U1,U2, {Star({k(U5,U1)}r'1, k(U1,U2)
)}k(U1,U2));
  claim_U1(U1, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
    }
role U2{
    fresh r2 : XOR;
    var T2-1, T2-3, T2-4, T2-5, t2 : Ticket;
    var r'1, r1, r3, r4, r5 : XOR;
    recv_1(U1,U2, {{T2-1}r'1}r1);
    send_5(U2,U1, {{k(U1,U2)}k(U2,U3)}r2 );
    send_6(U2,U3, {{k(U1,U2)}k(U2,U3)}r2 );
    send_7(U2,U4, {{k(U1,U2)}k(U2,U3)}r2 );
    send_8(U2,U5, {{k(U1,U2)}k(U2,U3)}r2 );
    recv_10(U3,U2, {T2-3}r3);
    recv_14(U4,U2, {T2-4}r4);
    recv_18(U5,U2, {T2-5}r5);
    recv_21(U1,U2, t2 );
       send_22(U2,U3, {Star({k(U5,U1)}r'1, k(U2,U3))}
k(U2,U3));
  claim_U2(U2, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
    }
role U3 {
    fresh r3 : XOR;
    var T3-1, T3-2, T3-4, T3-5, t3 : Ticket;
    var r'1, r1, r2, r4, r5 : XOR;
    recv_2(U1,U3,{{T3-1}r'1}r1 );
    recv_6(U2,U3, {T3-2}r2 );
    send_9(U3,U1, {{k(U2,U3)}k(U3,U4)}r3 );
    send_10(U3,U2, {{k(U2,U3)}k(U3,U4)}r3);
    send_11(U3,U4, {{k(U2,U3)}k(U3,U4)}r3);
    send_12(U3,U5, {{k(U2,U3)}k(U3,U4)}r3);
    recv_15(U4,U3, {T3-4}r4 );
    recv_19(U5,U3, {T3-5}r5 );
    recv_22(U2,U3, t3 );
  send_23(U3,U4, {Star({k(U5,U1)}r'1, k(U3,U4))}k(U3,U4));
  claim_U3(U3, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
    }
role U4 {
    fresh r4 : XOR;
    var T4-1, T4-2, T4-3, T4-5, t4 : Ticket;
    var r'1, r1, r2, r3, r5 : XOR;
    recv_3(U1,U4,{{T4-1}r'1}r1);
    recv_7(U2,U4, {T4-2}r2 );
    recv_11(U3,U4, {T4-3}r3);
    send_13(U4,U1, {{k(U3,U4)}k(U4,U5)}r4);
    send_14(U4,U2, {{k(U3,U4)}k(U4,U5)}r4);
    send_15(U4,U3, {{k(U3,U4)}k(U4,U5)}r4);
    send_16(U4,U5, {{k(U3,U4)}k(U4,U5)}r4);
    recv_20(U5,U4, {T4-5}r5);
    recv_23(U3,U4, t4);
  send_24(U4,U5, {Star({k(U5,U1)}r'1, k(U4,U5))}k(U4,U5));
  claim_U4(U4, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
    }
role U5 {
    fresh r5 : XOR;
    var T5-1, T5-2, T5-3, T5-4, t5 : Ticket;
    var r'1, r1, r2, r3, r4 : XOR;
    recv_4(U1,U5,{{T5-1}r'1}r1);
    recv_8(U2,U5, {T5-2}r2 );
    recv_12(U3,U5, {T5-3}r3 );
    recv_16(U4,U5, {T5-4}r4 );
    send_17(U5,U1, {{k(U4,U5)}k(U5,U1)}r5);
    send_18(U5,U2, {{k(U4,U5)}k(U5,U1)}r5);
    send_19(U5,U3, {{k(U4,U5)}k(U5,U1)}r5);
    recv_24(U4,U5, t5);
  claim_U5(U5, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
}
}
```

Appendix B

The Scyther verification code for secrecy and non injective synchronisation claim:

```
const Star: Function;
hashfunction H, hmac;
usertype XOR;
usertype Timestamp;
protocol OPR(U1,U2,U3,U4,U5){
role U1 {
    fresh r1 : XOR;
    fresh r'1 : XOR;
    fresh T1, T'1, T"1, T"'1, T""1 : Timestamp;
    var T2, T3, T4, T5 : Timestamp;
    var T1-2, T1-3, T1-4, T1-5 : Ticket;
    var r2, r3, r4, r5 : XOR;
  send_1(U1,U2, {{k(U1,U2)}r'1}r1, T1, U1, hmac(k(U1,U2),
        {{k(U1,U2)}r'1}r1, T1, U1));
  send_2(U1,U3, {{k(U1,U2)}r'1}r1, T'1, U1, hmac(k(U1,U3),
        {{k(U1,U2)}r'1}r1, T'1, U1));
    send_3(U1,U4, {{k(U1,U2)}r'1}r1, T"1, U1, hmac(k(U1,U4),
            {{k(U1,U2)}r'1}r1, T"1, U1));
            send_4(U1,U5, {{k(U1,U2)}r'1}r1, T"'1, U1,
hmac(k(U1,U5),
            {{k(U1,U2)}r'1}r1, T"'1, U1));
      recv_5(U2,U1, {T1-2}r2 , T2, U2, hmac( k(U2,U1),{T1-
2}r2,
            U2, T2));
      recv_9(U3,U1, {T1-3}r3, T3, U3, hmac(k(U3,U1), {T1-
3}r3,
           T3, U3));
      recv_13(U4,U1, {T1-4}r4, T4, U4, hmac(k(U4,U1),{T1-
4}r4,
           T4, U4));
        recv_17(U5,U1, {T1-5}r5, T5, U5, hmac(k(U5,U1),
{T1-5}r5,
           T5, U5));
    send_21(U1,U2, {Star({k(U5,U1)}r'1, k(U1,U2))}k(U1,U2),
           T""1, U1, hmac(k(U1,U2), {Star({k(U5,U1)}r'1,
           k(U1,U2))}k(U1,U2), T""1, U1));
    claim_U1(U1, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
     claim_U12(U1, Nisynch);
    }
role U2 {
      fresh r2 : XOR;
      fresh T2, T'2, T"2, T"'2, T""2 : Timestamp;
      var T1, T'3, T'4, T'5, T""1 : Timestamp;
      var T2-1, T2-3, T2-4, T2-5, t2 : Ticket;
      var r'1, r1, r3, r4, r5 : XOR;
      recv_1(U1,U2, {{T2-1}r'1}r1, T1, U1, hmac(k(U1,U2),
          {{T2-1}r'1}r1,T1, U1));
      send_5(U2,U1, {{k(U1,U2)}k(U2,U3)}r2, T2, U2,
          hmac(k(U2,U1),{{k(U1,U2)}k(U2,U3)}r2, U2, T2));
      send_6(U2,U3, {{k(U1,U2)}k(U2,U3)}r2, T'2, U2,
         hmac(k(U2,U3), {{k(U1,U2)}k(U2,U3)}r2, T'2, U2));
      send_7(U2,U4, {{k(U1,U2)}k(U2,U3)}r2, T"2, U2,
         hmac(k(U2,U4),{{k(U1,U2)}k(U2,U3)}r2, T"2, U2));
      send_8(U2,U5, {{k(U1,U2)}k(U2,U3)}r2, T"'2, U2,
         hmac(k(U2,U5),{{k(U1,U2)}k(U2,U3)}r2, T"'2, U2));
        recv_10(U3,U2, {T2-3}r3, T'3, U3, hmac(k(U3,U2),
{T2-3}r3,
           T'3, U3) );
        recv_14(U4,U2, {T2-4}r4, T'4, U4, hmac(k(U4,U2),
{T2-4}r4,
           T'4, U4) );
        recv_18(U5,U2, {T2-5}r5, T'5, U5, hmac(k(U5,U2),
{T2-5}r5,
           T'5, U5) );
        recv_21(U1,U2, t2, T""1, U1, hmac(k(U1,U2), t2 ,
T""1, U1));        send_22(U2,U3, {Star({k(U5,U1)}r'1,
k(U2,U3))}k(U2,U3),
           T""2, U2, hmac(k(U2,U3), {Star({k(U5,U1)}r'1,
           k(U2,U3))}k(U2,U3), T""2, U2));
   claim_U2(U2, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
    claim_U22(U2, Nisynch);
    }
role U3 {
    fresh r3 : XOR;
    fresh T3, T'3, T"3, T"'3, T""3 : Timestamp;
    var T'1, T'2, T"4, T"5, T""2 : Timestamp;
    var T3-1, T3-2, T3-4, T3-5, t3 : Ticket;
    var r'1, r1, r2, r4, r5 : XOR;
    recv_2(U1,U3,{{T3-1}r'1}r1, T'1, U1, hmac(k(U1,U3),
         {{T3-1}r'1}r1, T'1, U1));
    recv_6(U2,U3, {T3-2}r2, T'2, U2, hmac(k(U2,U3),
         {T3-2}r2, T'2, U2));
    send_9(U3,U1, {{k(U2,U3)}k(U3,U4)}r3, T3, U3,
        hmac(k(U3,U1),{{k(U2,U3)}k(U3,U4)}r3, T3, U3));
    send_10(U3,U2, {{k(U2,U3)}k(U3,U4)}r3, T'3, U3,
       hmac(k(U3,U2), {{k(U2,U3)}k(U3,U4)}r3, T'3, U3));
    send_11(U3,U4, {{k(U2,U3)}k(U3,U4)}r3, T"3, U3,
       hmac(k(U3,U4), {{k(U2,U3)}k(U3,U4)}r3, T"3, U3));
    send_12(U3,U5, {{k(U2,U3)}k(U3,U4)}r3, T"'3, U3,
       hmac(k(U3,U5), {{k(U2,U3)}k(U3,U4)}r3, T"'3, U3));
      recv_15(U4,U3, {T3-4}r4, T"4, U4, hmac(k(U4,U3),
{T3-4}r4,
          T"4, U4));
      recv_19(U5,U3, {T3-5}r5, T"5, U5, hmac(k(U5,U3),
{T3-5}r5,
          T"5, U5));
    recv_22(U2,U3, t3, T""2, U2, hmac(k(U2,U3),
        t3, T""2, U2));
   send_23(U3,U4, {Star({k(U5,U1)}r'1, k(U3,U4))}k(U3,U4),
         T""3, U3, hmac(k(U3,U4), {Star({k(U5,U1)}r'1,
         k(U3,U4))}k(U3,U4), T""3, U3));
  claim_U3(U3, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
    claim_U32(U3, Nisynch);
    }
role U4 {
    fresh r4 : XOR;
    fresh T4, T'4, T"4, T"'4, T""4 : Timestamp;
    var T"1, T"2, T"3, T"'5, T""3 : Timestamp;
    var T4-1, T4-2, T4-3, T4-5, t4 : Ticket;
    var r'1, r1, r2, r3, r5 : XOR;
    recv_3(U1,U4,{{T4-1}r'1}r1, T"1, U1, hmac(k(U1,U4),
        {{T4-1}r'1}r1, T"1, U1));
    recv_7(U2,U4, {T4-2}r2, T"2, U2, hmac(k(U2,U4),
        {T4-2}r2, T"2, U2));
    recv_11(U3,U4, {T4-3}r3, T"3, U3, hmac(k(U3,U4),
        {T4-3}r3, T"3, U3));
    send_13(U4,U1, {{k(U3,U4)}k(U4,U5)}r4, T4, U4,
       hmac(k(U4,U1), {{k(U3,U4)}k(U4,U5)}r4, T4, U4));
    send_14(U4,U2, {{k(U3,U4)}k(U4,U5)}r4, T'4, U4,
       hmac(k(U4,U2), {{k(U3,U4)}k(U4,U5)}r4, T'4, U4));
    send_15(U4,U3, {{k(U3,U4)}k(U4,U5)}r4, T"4, U4,
       hmac(k(U4,U3), {{k(U3,U4)}k(U4,U5)}r4, T"4, U4));
    send_16(U4,U5, {{k(U3,U4)}k(U4,U5)}r4, T"'4, U4,
       hmac(k(U4,U5), {{k(U3,U4)}k(U4,U5)}r4, T"'4, U4));
    recv_20(U5,U4, {T4-5}r5, T"'5, U5, hmac(k(U5,U4),
         {T4-5}r5, T"'5, U5));
    recv_23(U3,U4, t4 , T""3, U3, hmac(k(U3,U4), t4, T""3,
U3));
```

```
    send_24(U4,U5, {Star({k(U5,U1)}r'1, k(U4,U5))}k(U4,U5),
              T""4, U4, hmac(k(U4,U5), {Star(k(U5,U1)r'1,
              k(U4,U5))}k(U4,U5), T""4, U4));
   claim_U4(U4, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
     claim_U42(U4, Nisynch);
     }
role U5 {
     fresh r5 : XOR;
     fresh T5, T'5, T"5, T"'5, T""5 : Timestamp;
     var T"'1, T"'2, T"'3, T"'4, T""4 : Timestamp;
     var T5-1, T5-2, T5-3, T5-4, t5 : Ticket;
     var r'1, r1, r2, r3, r4 : XOR;
     recv_4(U1,U5,{{T5-1}r'1}r1, T"'1, U1, hmac(k(U1,U5),
               {{T5-1}r'1}r1, T"'1, U1) );
     recv_8(U2,U5, {T5-2}r2, T"'2, U2, hmac(k(U2,U5),
               {T5-2}r2, T"'2, U2) );
     recv_12(U3,U5, {T5-3}r3, T"'3, U3, hmac(k(U3,U5),
               {T5-3}r3, T"'3, U3) );
     recv_16(U4,U5, {T5-4}r4, T"'4, U4, hmac(k(U4,U5),
               {T5-4}r4, T"'4, U4));
     send_17(U5,U1, {{k(U4,U5)}k(U5,U1)}r5, T5, U5,
            hmac(k(U5,U1), {{k(U4,U5)}k(U5,U1)}r5, T5, U5));
     send_18(U5,U2, {{k(U4,U5)}k(U5,U1)}r5, T'5, U5,
           hmac(k(U5,U2), {{k(U4,U5)}k(U5,U1)}r5, T'5, U5));
     send_19(U5,U3, {{k(U4,U5)}k(U5,U1)}r5, T"5, U5,
           hmac(k(U5,U3), {{k(U4,U5)}k(U5,U1)}r5, T"5, U5));
     send_20(U5,U4, {{k(U4,U5)}k(U5,U1)}r5, T"'5, U5,
          hmac(k(U5,U4), {{k(U4,U5)}k(U5,U1)}r5, T"'5, U5));
     recv_24(U4,U5, t5, T""4, U4, hmac(k(U4,U5),
               t5, T""4, U4));
   claim_U5(U5, Secret, H({k(U5,U1)}r'1, {{{{r5}r4}r3}r2}r1));
     claim_U52(U5, Nisynch);
     }
}
```

Hossein Oraei was born in 1990 in Isfahan. He received the B.Sc. and M.Sc. degrees with honors in Mathematics from Qom University and Sharif University of Technology in 2012 and 2014, respectively. He is currently a Ph.D. student in the School of Mathematical sciences at Iran Universityof Science and Technology. His research interests include Symmetric Cryptography and Secret Sharing schemes.

Mohsen Pourpouneh was born in 1989 in Isfahan. He got his B.S. and M.S. in Computer Science, from Shahid Beheshti University and Tehran University, respectively. He started his career as a Ph.D. student at Sharif University of Technology. His research interest includes formal method, electronic voting protocols, and multi-agent systems.

Rasoul Ramezanian was born in Mashhad in 1979. He got his B.S. and M.S. in Mathematics. In 2008, he graduated from a Ph.D. program at Mathematical Sciences Department of Sharif University of Technology. He is an assistant professor at the faculty of Mathematical Sciences at Ferdowsi University of Mashhad. His research interests include formal method, specifying and verifying security protocols, multi-agent systems, and process algebra.