symmetry

MDPI

Article

Cryptanalysis of a Group Key Establishment Protocol

Jorge Martínez Carracedo [1,†] and Adriana Suárez Corona [2,*,†]

[1] School of Computing and Mathematics, Ulster University, Belfast BT37 0QB, UK; j.martinez-carracedo@ulster.ac.uk
[2] Department of Mathematical Sciences, Universidad de León, 24071 León, Spain
* Correspondence: asuac@unileon.es
† These authors contributed equally to this work.

Abstract: In this paper, we analyze the security of a group key establishment scheme proposed by López-Ramos et al. This proposal aims at allowing a group of users to agree on a common key. We present several attacks against the security of the proposed protocol. In particular, an active attack is presented, and it is also proved that the protocol does not provide forward secrecy.

Keywords: cryptanalysis; group key establishment

Citation: Carracedo, J.M.; Corona, A.S. Cryptanalysis of a Group Key Establishment Protocol. *Symmetry* **2021**, *13*, 332. https://doi.org/10.3390/sym13020332

Academic Editor: Juan Alberto Rodríguez Velázquez
Received: 20 January 2021
Accepted: 9 February 2021
Published: 17 February 2021

1. Introduction

Secure multiparty communication is an important concern for many current applications that work over public insecure channels, such as the Internet. Wireless sensor networks, collaborative applications, multiparty voice and video conferences, etc. need to guarantee confidentiality, integrity and authentication in their communications.

Group key establishment (GKE) protocols are fundamental in that sense. They allow a set of participants to agree on a common secret key to be used afterwards with symmetric key cryptographic primitives.

In some settings all the nodes play an equivalent role, and thus the group protocol is somewhat symmetric. Nevertheless, there are other applications where some nodes are distinguished and one can assume they may have more computational power and resources, and thus, they are required to perform more computations.

Over recent decades, group key establishment protocols were widely discussed in the literature [1–7], and formal security models were proposed, indicating which attacks the adversary can perform and what a secure key establishment protocol is. What is typically required is that, after completion of the protocol, the intended users agree on a common key, whereas the adversary does not learn anything about it.

A standard technique to augment the security of a scheme is the use of compilers, which allows a modular design, going from passively secure solutions to authenticated ones [8], from 2-party to group solutions [9], or adding forward secrecy [10].

However, several protocols were found to be insecure after they were published, because the proposals do not provide security proofs or the proofs are not correct [11–13]. Other protocols were found to be insecure when considering active attacks [14].

Motivated by the works in López-Ramos et al. [14], in this paper, we analyze a group key establishment proposal by López-Ramos et al. [15] and present several attacks on the proposed protocols. In particular, we present here some active attacks against the protocols, proving they are insecure when considering active adversaries.

Contributions: We present several concrete attacks showing the security flaws of the protocols proposed in López-Ramos et al. [15]. In Section 2, we review the proposal of López Ramos et al. Then, in Section 3 we review a standard security model for group key exchange. We then present our attacks in Section 4.

2. The Protocol of López Ramos et al.

In this section, we describe Protocol 1 in López-Ramos et al. [15], which can be seen as an extension of the classical 2-party Diffie-Hellman key exchange. Four different protocols are presented, which are modifications of this first one. In particular, Protocol 2 computes the same session key, but publishing only one public key and sending a different message in Round 2. Protocol 3 describe the extra steps to be done if some participants leave the group and Protocol 4 deals with the case where some users join the group.

Initialization

Let $\{U_1, \ldots, U_n\}$ be the finite set of protocol participants, including U_{c_1}, who will act as controller. The users agree on a multiplicative cyclic group G of prime order p and on g, a generator of G.

Each user $U_i, 1 \leq i \leq n$ will have two random values, $r_i, x_i \in \mathbb{Z}_p^*$ as private keys and g^{r_i} and g^{x_i} will be their public keys.

Round 1

1. Each user U_i publishes his pair of public keys (g^{r_i}, g^{x_i}) (We assume that these keys are sent to the users, hence the adversary can potentially manipulate those values).
2. The group controller calculate $K_1 = g^{r_{c_1} \sum_{j=1, j \neq c_1}^{n} r_j}$, which will be the session key.
3. The group controller will choose a new pair of elements (r'_{c_1}, x'_{c_1}) that will be privately kept and will become his new private information at a later stage.

Round 2

Every user U_i, using the public information, computes $g^{\sum_{j \neq i, c_1} r_j}$ and sends this value to U_{c_1} (Notice that there is no need to send this information, since this value can be computed from the published public keys).

The group controller U_{c_1}, moreover, computes

$$Y_{1,i} = g^{-x_{c_1} x_i} \left(g^{r_{c_1} \sum_{j \neq c_1, i} r_j} \right) \quad \text{for} \quad i \in \{1, \ldots, n\} \setminus \{c_1\} \quad \text{and}$$

$$Y_{1,c_1} = K_1 g^{-r'_{c_1} r_{c_1}} g^{-x'_{c_1} x_{c_1}},$$

$$R_1 = g^{r_{c_1}} \quad \text{and} \quad S_1 = g^{x_{c_1}}.$$

He broadcasts $(Y_{1,1}, \ldots, Y_{1,c_1}, \ldots, Y_{1,n}, R_1, S_1)$

Key Computation

Once user U_i has received the second round message, he computes the common session key $K_1 := K_{1,i} = Y_{1,i} S_1^{x_i} R_1^{r_i}$.

The protocol is summarized in Figure 1.

Remark 1. *The subindex* 1 *in the session key* K_1 *indicates here that it is the first execution of the protocol. In Protocols 3 and 4 in López-Ramos et al. [15], this subindex changes when the participants involved in the protocol change, i.e., some participants leave or join the protocol, and thus, some extra computations are needed.*

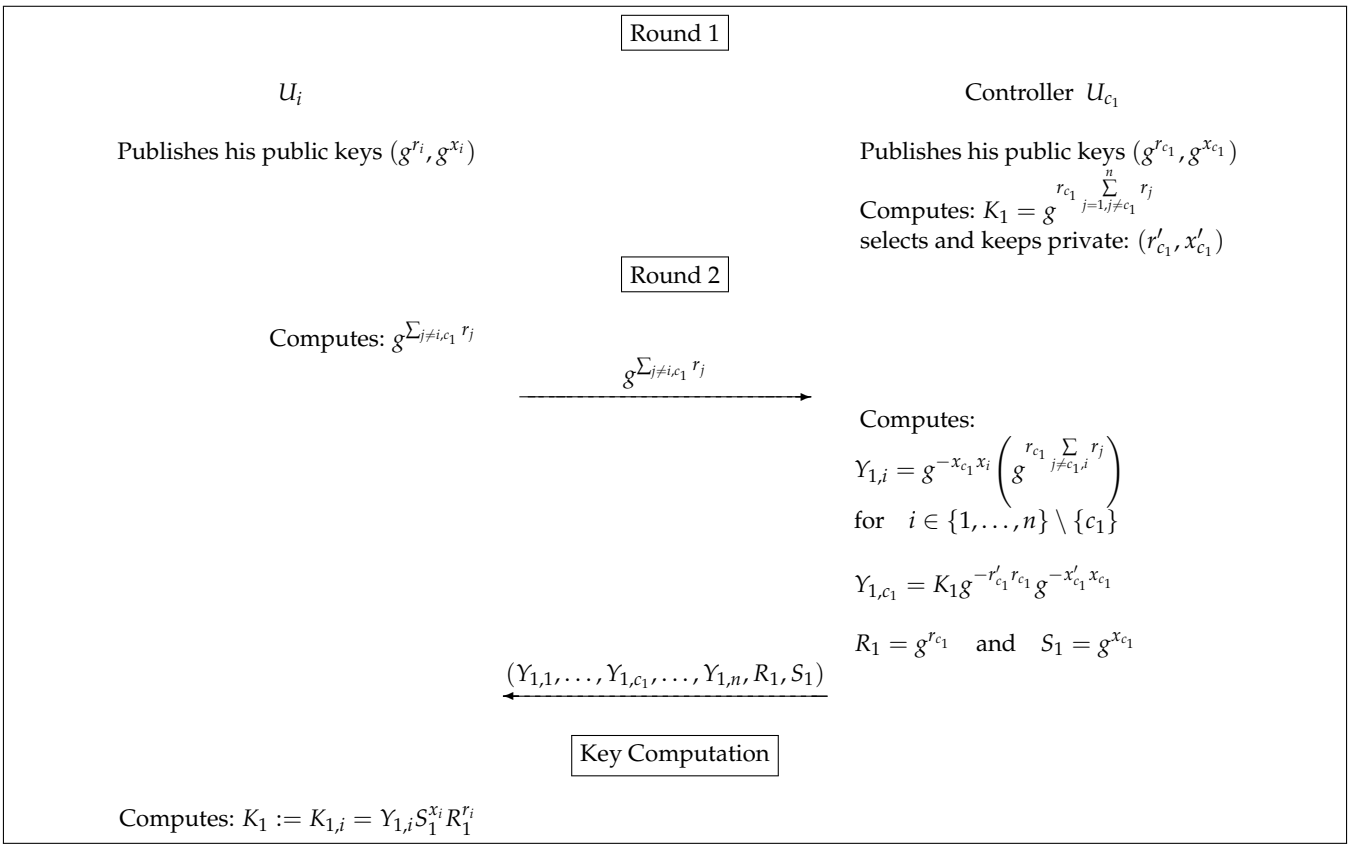

Figure 1. Protocol 1 of López Ramos et al.

3. Security Model

To formalize secure group key establishment, we use the somewhat standard Bohli et al.'s [5] security model, which builds on Jonathan Katz and Moti Yung [8].

Security Goals: Semantic Security and Authentication

Participants:

The (potential) protocol participants are modelled as probabilistic polynomial time (ppt) Turing machines in the finite set $\mathcal{U} = \{U_1, \ldots, U_n\}$. Each participant U_i in the set $\mathcal{U}$ is able to run a polynomial amount of protocol instances in parallel.

We will refer to instance s_i of principal U_i as $\Pi_i^{s_i}$ $(i \in \mathbb{N})$ and it has the following variables assigned:

$\mathsf{pid}_i^{s_i}$**:** stores the identities of the parties user U_i aims at establishing a session key with (including U_i itself);

$\mathsf{sid}_i^{s_i}$**:** is a variable storing a non-secret session identifier to the session key stored in $\mathsf{sk}_i^{s_i}$;

$\mathsf{acc}_i^{s_i}$**:** is a variable which indicates whether the session key in $\mathsf{sk}_i^{s_i}$ was accepted;

$\mathsf{term}_i^{s_i}$**:** is a variable which indicates whether the protocol execution has terminated;

$\mathsf{used}_i^{s_i}$**:** is a variable which indicates whether this instance is taking part in a protocol run;

$\mathsf{sk}_i^{s_i}$**:** this variable is initialized with a distinguished NULL value and will store the session key.

Communication network and adversarial capabilities:

We assume there exist arbitrary point to point connections among users and the network is non-private, fully asynchronous and in complete control of the adversary $\mathcal{A}$, who can eavesdrop, delay, delete, modify or insert messages. The adversary's capabilities are captured by the following *oracles*:

$\mathsf{Send}(U_i, s_i, M)$: when querying this oracle, message M is sent to instance $\Pi_i^{s_i}$ of user $U_i \in \mathcal{U}$. The output will be the protocol message that the instance outputs after receiving message M. This oracle can also be used for the adversary $\mathcal{A}$ to initialize a protocol execution, by using the special message $M = \{U_{i_1}, \ldots, U_{i_r}\}$ to an unused instance $\Pi_i^{s_i}$. This oracle initializes a protocol run among $U_{i_1}, \ldots, U_{i_r} \in \mathcal{U}$. After such a query, $\Pi_i^{s_i}$ sets $\mathsf{pid}_i^{s_i} := \{U_{i_1}, \ldots, U_{i_r}\}$, $\mathsf{used}_i^{s_i} :=$ TRUE, and processes the first step of the protocol.

$\mathsf{Execute}(U_1, s_1, \ldots, U_r, s_r)$: if the instances $s_1, \ldots, s_r$ have not yet been used, this oracle will return a transcript of a complete execution of the protocol among the specified instances.

$\mathsf{Reveal}(U_i, s_i)$: this oracle returns the session key stored in $sk_i^{s_i}$ if $\mathsf{acc}_i^{s_i} =$ TRUE and a NULL value otherwise.

$\mathsf{Corrupt}(U_i)$: this query returns U_i's long term secret key.

We can distinguish two types of adversaries. An adversary with access to all the oracles described above is considered to be *active*. If the adversary is not granted access to any of the Send oracles, then it is considered a *passive* adversary.

To define semantic security, we also allow the adversary to have access to a Test oracle, which can be queried only once. The query $\mathsf{Test}(U_i, s_i)$ can be made on input an instance $\Pi_i^{s_i}$ of user $U_i \in \mathcal{U}$ only if $\mathsf{acc}_i^{s_i} =$ TRUE. In that case, a bit $b \leftarrow \{0, 1\}$ is chosen uniformly at random; if $b = 0$, the oracle returns the session key stored in $\mathsf{sk}_i^{s_i}$. Otherwise, the oracle outputs a uniformly at random chosen element from the space of session keys.

Security notions:

For the schemes to be useful, we need the group key establishments to be *correct*, i.e., without adversarial interference, the protocol would allow all users to compute the same key.

Definition 1 (correctness)**.** *A group key establishment is* correct *if for all instances* $\Pi_i^{s_i}$, $\Pi_j^{s_j}$ *which accepted with* $\mathsf{sid}_i^{s_i} = \mathsf{sid}_j^{s_j}$ *and* $\mathsf{pid}_i^{s_i} = \mathsf{pid}_j^{s_j}$, *the condition* $\mathsf{sk}_i^{s_i} = \mathsf{sk}_j^{s_j} \neq$ NULL *is satisfied.*

To be more precise in the security definition, it is important to specify under which conditions the adversary can query the Test oracle. To do so, we first define the following notion of *partnering*:

Definition 2 (partnering)**.** *Two terminated instances* $\Pi_i^{s_i}$ *and* $\Pi_j^{s_j}$ *are* partnered *if* $\mathsf{sid}_i^{s_i} = \mathsf{sid}_j^{s_j}$, $\mathsf{pid}_i^{s_i} = \mathsf{pid}_j^{s_j}$ *and* $\mathsf{acc}_{U_i}^{s_i} = \mathsf{acc}_{U_j}^{s_j} =$ TRUE.

To avoid queries that would trivially allow the adversary to know the key, we restrict the instances that can be queried to the Test oracle, only allowing *fresh* instances:

Definition 3 (freshness)**.** *We say an instance* $\Pi_i^{s_i}$ *is* fresh *if none of the following events has occurred:*

- *the adversary queried* $\mathsf{Reveal}(U_j, s_j)$ *for an instance* $\Pi_j^{s_j}$ *that is partnered with* $\Pi_i^{s_i}$;
- *the adversary queried* $\mathsf{Corrupt}(U_j)$ *for a user* $U_j \in \mathsf{pid}_i^{s_i}$ *before a query of the form* $\mathsf{Send}(U_l, s_l, *)$;

Remark 2. *The previous definition for freshness allows including the desired goal of* forward secrecy *in our definition of security given below: an adversary* $\mathcal{A}$ *is allowed to query* Corrupt *for* all *users and obtain their long term keys without violating freshness, if he does not send any message afterwards.*

Let $\mathsf{Succ}_{\mathcal{A}}$ be the event that the adversary $\mathcal{A}$ queries the Test oracle with a fresh instance and makes a correct guess about the random bit b used by the Test oracle, we define the *advantage* of an adversary $\mathcal{A}$ attacking protocol P as

$$\mathrm{Adv}^{\mathrm{ke}}_{\mathcal{A}} = \mathrm{Adv}^{\mathrm{ke}}_{\mathcal{A}}(k) := \left|\Pr[\mathsf{Succ}_{\mathcal{A}}] - \frac{1}{2}\right|.$$

Definition 4 (semantic security)**.** *A group key establishment protocol is* (semantically) secure, *if* $\mathrm{Adv}^{\mathrm{ke}}_{\mathcal{A}} = \mathrm{Adv}^{\mathrm{ke}}_{\mathcal{A}}(k)$ *is negligible for every ppt adversary* $\mathcal{A}$*.*

4. Cryptanalysis of the Proposal of López-Ramos et al.

In this section, we describe several concrete attacks refuting the security results of López-Ramos et al. [15], where four different, but related, GKE protocols are described. The four protocols will be considered in this section. However, we will only explicitly attack Protocol 1, being the attacks to the others straightforwardly adapted.

4.1. Active Attack

Informally, since the protocol is not authenticated, we will describe here how an adversary can attack the protocol by mounting a Man-In-The-Middle attack. Users will end up sharing a key with the adversary, instead of with all the intended communication partners. We formalize the attack below.

Let us fix $\{U_1, ..., U_n\}$ the set of communication parties and let $\mathcal{A}$ be an active attacker able to supersede some parties in the set. We will distinguish two different cases: $\mathcal{A}$ shares a key with the group controller U_{c_1} and other with the rest of the users and $\mathcal{A}$ shares a key with any other party U_i, $i \neq c_1$, and a different key with the rest, including the controller.

If $\mathcal{A}$ tries to share a different key with the group controller U_{c_1} the adversary can build an attack by following the next steps:

1. The attacker $\mathcal{A}$ queries $\mathsf{Send}(U_1, s_1, \ldots, U_n, s_n)$, to initiate a protocol instance. After this query, the first step of the protocol is executed. In particular, the adversary obtains every users' pairs of public keys (g^{r_i}, g^{x_i}), with $r_i, x_i \in \mathbb{Z}_p^*$.
2. The adversary $\mathcal{A}$ will delete the message $(g^{r_{c_1}}, g^{x_{c_1}})$ sent by the controller U_{c_1} to the rest of the users and delete the public keys (g^{r_1}, g^{x_1}) sent by user U_1 to U_{c_1}.
3. The adversary $\mathcal{A}$ generates its private keys $a_{c_1}, b_{c_1} \in \mathbb{Z}_p^*$ and public keys $(g^{a_{c_1}}, g^{b_{c_1}})$ and queries $\mathsf{Send}(U_i, s_i, (g^{a_{c_1}}, g^{b_{c_1}}))$, for all $i \in \{1, ..., n\} \setminus \{c_1\}$. the adversary $\mathcal{A}$ generates its private keys $a_1, b_1 \in \mathbb{Z}_p^*$ and public keys (g^{a_1}, g^{b_1}) and queries $\mathsf{Send}(U_{c_1}, s_{c_1}, (g^{a_1}, g^{b_1}))$.

   Notice that every user U_i, $i \neq 1, c_1$, after receiving that message, will compute and send the value $g^{\sum_{j=1, j\neq c_1}^{n} r_j}$ and therefore this value will be output by the Send oracle.

   The controller U_{c_1}, after receiving that message, will compute and send the value $g^{a_1 + \sum_{j=2, j\neq c_1}^{n} r_j}$ and therefore this value will be output by the Send oracle.

4. The adversary $\mathcal{A}$ will compute the session key $Q_1 = g^{a_{c_1}(\sum_{j=1, j\neq c_1}^{n} r_j)}$ and the values $T_1 = g^{a_{c_1}}$ and $V_1 = g^{b_{c_1}}$, along with the keying values

$$Z_{1,i} = g^{-b_{c_1} x_i} g^{a_{c_1}(\sum_{j=1, j\neq c_1, i}^{n} r_j)}, \quad i \neq c_1$$

$$Z_{1,c_1} = Q_1 g^{-a'_{c_1} a_{c_1}} g^{-b'_{c_1} b_{c_1}}.$$

5. The adversary $\mathcal{A}$ will query $\mathsf{Send}(U_i, s_i, (Z_{1,1}, \ldots, Z_{1,n}, T_1, V_1)$ oracle, for all $i \in \{1, ..., n\} \setminus \{c_1\}$.

6. The adversary $\mathcal{A}$ will compute the session key $K_1 = g^{r_{c_1}(\sum_{j=1,j\neq c_1}^{n} r_j)}$ and the values $R_1 = g^{r_{c_1}}$ and $S_1 = g^{x_{c_1}}$, along with the keying values

$$Y_{1,i} = g^{-x_{c_1}x_i} g^{r_{c_1}(a_1 + \sum_{j=2,j\neq c_1,i}^{n} r_j)}, \quad i \neq c_1$$

$$Y_{1,c_1} = K_1 g^{-r'_{c_1} r_{c_1}} g^{-x'_{c_1} x_{c_1}}.$$

7. The adversary $\mathcal{A}$ will query $\mathsf{Send}(U_1, s_1, (Y_{1,1}, \ldots, Y_{1,n}, S_1, T_1)$ oracle.

Please note that after receiving this last message, users $\{U_1, \ldots, U_n\} \setminus \{U_{c_1}\}$, following the protocol, will compute $Q_{1,i} = Z_{1,i} T_1^{x_i} V_1^{r_i}$. Please note that $Q_{1,i} = Q_1$ for every $i \neq c_1$.

On the other hand, the group controller U_{c_1} will compute $K_1 = Y_{1,1} S_1^{b_1} R_1^{a_1}$.

Therefore, after this attack, the adversary has established a shared key Q_1 with the set of parties $\{U_1, \ldots, U_n\} \setminus \{U_{c_1}\}$ and the key K_1 with the group controller U_{c_1}, where

$$Q_1 = g^{a_{c_1} \sum_{j=1,j\neq c_1}^{n} r_j} \quad \text{and} \quad K_1 = g^{r_{c_1}(a_1 + \sum_{j=2,j\neq c_1}^{n} r_j)}.$$

Consequently, all the users will believe they are establishing a common key when they are not. Moreover, the adversary can decrypt the messages sent encrypted with both keys and forward the communication between the users that do not share a key.

This attack is outlined in Figure 2.

If $\mathcal{A}$ tries to compute a different key with any user different from the group controller, we can assume without loss of generality that $\mathcal{A}$ is sharing it with U_1. The adversary $\mathcal{A}$ can build an attack following the subsequent steps:

1. The attacker $\mathcal{A}$ queries $\mathsf{Send}(U_1, s_1, \ldots, U_n, s_n)$, to initiate a protocol instance. After this query, the first step of the protocol is executed. In particular, the users send their public keys and thus, the adversary obtains (g^{r_i}, g^{x_i}), with $r_i, x_i \in \mathbb{Z}_p{}^*$ for all the participants $\{U_1, ..., U_n\}$.
2. The adversary $\mathcal{A}$ will delete the message $(g^{r_{c_1}}, g^{x_{c_1}})$ sent by the controller U_{c_1} to user U_1 and the message (g^{r_1}, g^{x_1}) sent by user U_1 to the rest of the participants.
3. The adversary $\mathcal{A}$, will choose random values $a_1, b_1, a_{c_1}, b_{c_1} \in \mathbb{Z}_p{}^*$, and queries $\mathsf{Send}(U_i, s_i, (g^{a_1}, g^{b_1}))$, for all $i \in \{2, ..., n\}$, including c_1.

Notice that every user U_i, $i \neq 1, c_1$ and the adversary $\mathcal{A}$, after receiving that message, will compute $g^{(a_1 + \sum_{j=2,j\neq c_1}^{n} r_j)}$ and therefore this value will be output by the Send oracle.

Moreover, the group controller U_{c_1} will calculate the session key $Q_1 = g^{r_{c_1}(a_1 + \sum_{j=2,j\neq c_1}^{n} r_j)}$ and he will send $R_1 = g^{r_{c_1}}$ and $S_1 = g^{x_{c_1}}$, along with the keying values

$$Z_{1,1} = g^{-x_{c_1} b_1} g^{(r_{c_1} \sum_{j=2,j\neq c_1}^{n} r_j)},$$

$$Z_{1,i} = g^{-x_{c_1}x_i} g^{r_{c_1}(a_1 + \sum_{j=2,j\neq c_1}^{n} r_j)} \quad i \neq 1, c_1,$$

$$Z_{1,c_1} = Q_1 g^{-r'_{c_1} r_{c_1}} g^{-x'_{c_1} x_{c_1}}.$$

These values will also be part of the output of the Send oracle.

Please note that after receiving this message every user U_i, $i \neq 1$, can compute the key $Q_1 = Z_{1,i} S_1^{x_i} R_1^{r_i}$ that will be shared with the adversary $\mathcal{A}$.

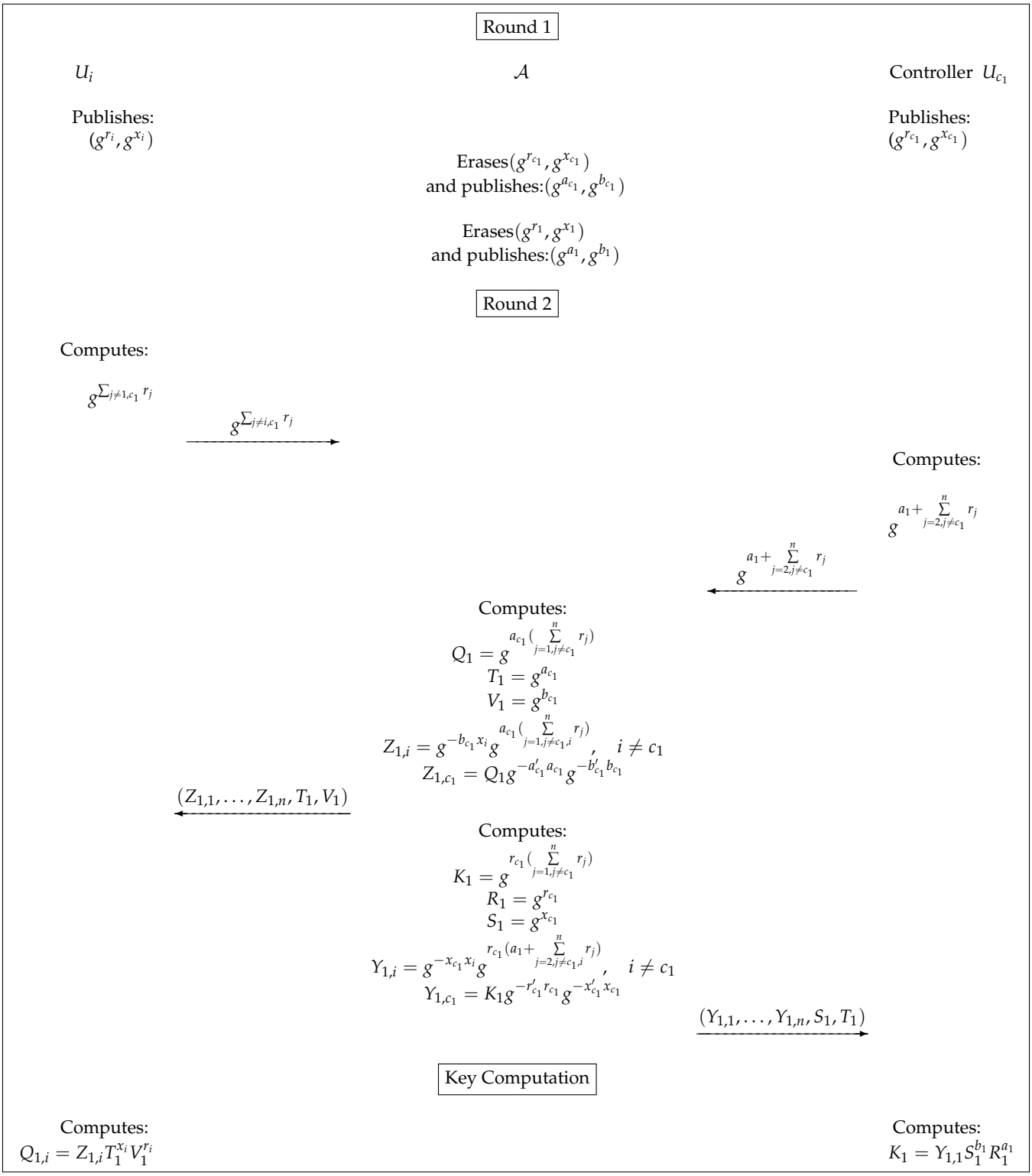

Figure 2. Active attack on Protocol 1 of López Ramos et al.

4. The attacker $\mathcal{A}$ will delete the message sent by U_{c_1} to the superseded user U_1, and queries $\mathsf{Send}(U_1, s_1, (W_{1,1}, \ldots, W_{1,n}, T_1, V_1))$, where

$$W_{1,i} = g^{-b_{c_1}x_i} g^{a_{c_1}(\sum_{j=1,j\neq i,c_1}^{n} r_j)},$$

$$W_{1,c_1} = K_1 g^{-a'_{c_1}a_{c_1}} g^{-b'_{c_1}b_{c_1}},$$

$$T_1 = g^{a_{c_1}} \quad \text{and} \quad V_1 = g^{b_{c_1}}.$$

Please note that user U_1, after receiving these last messages, can compute the key $K_1 = W_{1,1} T_1^{x_1} V_1^{r_1}$ which is shared with the adversary $\mathcal{A}$.

5. With the information received, the users, following the protocol, will compute the subsequent keys:
   (a) The superseded user U_1 will compute $K_1 = W_{1,1} T_1^{x_1} V_1^{r_1}$.
   (b) Every user U_i, $i \neq 1$ computes $Q_{1,i} = Y_{1,i} S_1^{x_i} R_1^{r_i}$.
   (c) Adversary $\mathcal{A}$ computes $Q_{1,1} = Z_{1,1} S_1^{b_1} R_1^{a_1}$ and $K_1 = W_{1,1} T_1^{x_1} V_1^{r_1}$.

Therefore, the adversary $\mathcal{A}$ has established a shared key Q_1 with the set of parties $\{U_2, ..., U_n\}$. On the other hand, both U_1 and the adversary $\mathcal{A}$ share the common key K_1.

Remark 3. *While in López-Ramos et al. [15] four different protocols were described, in the previous lines only Protocol 1 was attacked.*

In Protocol 2, authors try to share the computational requirements in a more even way among the parties by slightly modifying which values every participant sends to the group controller and the computations that this user has to perform. However, the only private information for every user is the tuple (r_i, x_i) as in Protocol 1. Thus, an attack can be built analogously by following the steps described above.

In Protocol 3, authors assume that the group controller has changed. The new group controller, by using two private elements (r'_{c_t}, x'_{c_t}) makes a transformation of the key. The next steps of Protocol 3 follows the description of Protocol 1. Therefore, an attack can be built following the previous description.

In Protocol 4, new users take part in the round with new private elements (r_t, x_t). Therefore a new key has to be computed by the group controller using those new elements. Once more, subsequent steps of Protocol 4 follows the description of Protocol 1 and an attack can be constructed analogously.

4.2. Forward Secrecy

We will informally describe how a passive adversary who corrupts a participant $U_i \in \{U_1, \ldots, U_n\}$ involved in a protocol run will be able to compute the shared session key. Therefore, the protocol does not provide forward secrecy.

Let $\mathcal{A}$ be a probabilistic polynomial time adversary (modelled as a Turing machine). He may perform an attack by following the next steps:

1. The attacker $\mathcal{A}$ queries $\mathsf{Corrupt}(U_i)$, obtaining the private keys r_i and x_i.
2. Afterwards, he queries, $\mathsf{Execute}(U_1, s_1, \ldots, U_r, s_r)$, obtaining a protocol transcript. In particular, he gets the values $Y_{1,i}$, R_1 and S_1.
3. The adversary now can compute the key as user U_i would do according to the protocol description: $K_1 = Y_{1,i} S_1^{x_i} R_1^{r_i}$.
4. The adversary now queries $\mathsf{Test}(\mathsf{U_j}, \mathsf{s_j})$ on any user instance involved in the above execution. Since he knows the key established, he wins the game with probability one.

Please note that session s_j of user U_j remains fresh, since, the adversary has not made any Send or Reveal query, so the attack is legitimate.

Remark 4. *In Protocols 3 and 4 in López-Ramos et al. [15], it is described how to proceed when participants may join or leave the group. However, when a participant leaves, the only user changing his private and public keys is the new controller. This means that the rest of the users will have the same private and public key used for previous instances. Therefore, when corrupting any user that is not the new controller, one will obtain their private keys and mount the attack described above. Protocol 2, can also be attacked in the same way, just changing the computations to obtain the session key according to the protocol description.*

Remark 5. *As observed in Theorem 2.4 in López-Ramos et al. [15], the keying messages sent to establish the key can be seen as ElGamal-like encryptions of the key K_1 under a different key for*

each user. In that sense, the protocol can be interpreted as a key transport protocol, which cannot be forward secret.

Remark 6. ***Countermeasures:*** *If a security proof of the protocols in López-Ramos et al. [15] is provided for passive adversaries, and one consider the private keys as random nonces to be used only in one instance of the protocol, one could then apply the compiler in Katz and Yung [8] to avoid active attacks, generating long term keys for each user to compute digital signatures on all the exchanged messages to guarantee authentication. In that case, if the keys are nonces,* Corrupt *oracle queries would return the signing private keys, thus, corrupted users would not be able to compute the session keys and forward secrecy would also be granted.*

5. Conclusions

As demonstrated above, the protocol proposed by López Ramos et al. [15] does not offer security guarantees. The paper does not provide a rigorous security proof in any standard security model using provable security techniques. The proofs provided are too schematic. If a compiler for authentication is used and the private keys are ephemeral, some attacks could not be applicable. Nevertheless, a security proof should be provided.

Author Contributions: Individual contributions to this article: conceptualization, J.M.C. and A.S.C.; methodology, J.M.C. and A.S.C.; validation, J.M.C. and A.S.C.; formal analysis, J.M.C. and A.S.C.; software, J.M.C. and A.S.C.; investigation, J.M.C. and A.S.C.; resources, J.M.C. and A.S.C.; writing—original draft preparation, J.M.C. and A.S.C.; writing–review and editing, J.M.C. and A.S.C.; project administration, J.M.C. and A.S.C.; funding acquisition, J.M.C. and A.S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part through research project MTM2017-83506-C2-2-P by the Spanish MICINN.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bellare, M.; Rogaway, P. *Entity Authentication and Key Distribution*; CRYPTO, Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1993; Volume 773, pp. 232–249.
2. Bellare, M.; Pointcheval, D.; Rogaway, P. *Authenticated Key Exchange Secure against Dictionary Attacks*; EUROCRYPT, Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2000; Volume 1807, pp. 139–155.
3. Boyd, C.; Mathuria, A. *Protocols for Authentication and Key Establishment*; Information Security and Cryptography; Springer: Berlin/Heidelberg, Germany, 2003.
4. Burmester, M.; Desmedt, Y. A secure and scalable Group Key Exchange system. *Inf. Process. Lett.* **2005**, *94*, 137–143.
5. Bohli, J.; Vasco, M.I.G.; Steinwandt, R. Secure group key establishment revisited. *Int. J. Inf. Sec.* **2007**, *6*, 243–254, doi:10.1007/s10207-007-0018-x.
6. Boyd, C.; Davies, G.T.; Gjøsteen, K.; Jiang, Y. *Offline Assisted Group Key Exchange*; ISC, Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2018; Volume 11060, pp. 268–285.
7. Vasco, M.I.G.; del Pozo, A.L.P.; Corona, A.S. Group key exchange protocols withstanding ephemeral-key reveals. *IET Inf. Secur.* **2018**, *12*, 79–86.
8. Katz, J.; Yung, M. *Scalable Protocols for Authenticated Group Key Exchange*; CRYPTO, Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2729, pp. 110–125.
9. Abdalla, M.; Bohli, J.; Vasco, M.I.G.; Steinwandt, R. *(Password) Authenticated Key Establishment: From 2-Party to Group*; TCC, Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4392, pp. 499–514.
10. Neupane, K.; Steinwandt, R.; Corona, A.S. *Group Key Establishment: Adding Perfect Forward Secrecy at the Cost of One Round*; CANS; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7712, pp. 158–168.
11. Vasco, M.I.G.; Robinson, A.; Steinwandt, R. Cryptanalysis of a Proposal Based on the Discrete Logarithm Problem Inside S_n. *Cryptography* **2018**, *2*, 16, doi:10.3390/cryptography2030016.
12. Steinwandt, R.; Corona, A.S. Cryptanalysis of a 2-party key establishment based on a semigroup action problem. *Adv. Math. Commun.* **2011**, *5*, 87–92, doi:10.3934/amc.2011.5.87.

13. Vasco, M.I.G.; del Pozo, A.L.P.; Corona, A.S. Pitfalls in a server-aided authenticated group key establishment. *Inf. Sci.* **2016**, *363*, 1–7, doi:10.1016/j.ins.2016.05.004.
14. Baouch, M.; López-Ramos, J.A.; Torrecillas, B.; Schnyder, R. An active attack on a distributed Group Key Exchange system. *Adv. Math. Commun.* **2017**, *11*, 715–717.
15. López-Ramos, J.A.; Rosenthal, J.; Schipani, D.; Schnyder, R. An Application of Group Theory in Confidential Network Communications. *Math. Methods Appl. Sci.* **2016**, doi:10.1002/mma.4244.