# Multi-LHL protocol

Marika Mitrengová

Faculty of Mathematics, Physics and Informatics Comenius University, Mlynska dolina, 842 48 Bratislava, Slovakia
mitrengova@dcs.fmph.uniba.sk

**Abstract.** We present a password-authenticated group key exchange protocol where each user has his/her own password. Advantage of such protocol is in short passwords, which can be easily memorized. On the other hand these protocols face the low password entropy. In the first part we define security model based on models of Abdalla, Fouque and Pointcheval and Bellare, Pointcheval, Rogaway. We construct MLHL (Multi-LHL) protocol, which is based on LHL protocol proposed by Lee, Hwang and Lee. However, LHL protocol is flawed as pointed by Abdalla, Bresson, Chevassut and Choo, Raymond. We prove that our protocol is secure authenticated key exchange protocol with forward secrecy property and that the protocol is resistant against attacks on LHL protocol.

**Keywords:** protocol, password, security

## 1 Introduction

With the explosion of its size, Internet became a major communication channel among people. However, in its basis, Internet is an inherently insecure channel. The essential part of securing such channel is an exchange of cryptographically strong keys. People are notoriously bad at remembering long (pseudo)random sequences and thus the classical solution is to store the key on some device (*e.g.* hard disk, smart card) and protect it with a user password. This is inconvenient because the medium holding the original key needs to be carried everywhere by the user.

*Password authenticated key exchange (PAKE)* protocols were designed to alleviate this issue. They require a human user to remember only a short (easily-memorable) secret password. This is a major advantage for mobile users who need to authenticate at various places. PAKE protocols are therefore an interesting alternative of public key cryptography (PKI), especially in environments where PKI is hard to deploy. Because of their ability to distill low-quality user passwords to strong keys, PAKE protocols have received a lot of attention [14, 15, 18, 19].

Although the original idea of PAKE protocol EKE [1] was designed only for two participants, PAKE protocols can be used to authenticate multiple parties as well. The most important requirement is to require only a single password for the user. Solutions, where user has to remember one password per group of

participants obviously does not scale with human memory. Moreover, in case when one of the participants is compromised the whole group needs to choose a new password. Instead, the schemes with a single password per user offer much better user experience. This, however, comes at the cost of incorporating one party which will be trusted by everyone – a trusted server.

*Security issues with PAKE protocols:* As opposed to other cryptographic schemes, PAKE protocols contain one weak link in their security and that is the user password. Therefore, they must be guarded from a dictionary attack against a known dictionary $\mathcal{DICT}$ of all possible passwords. The dictionary attack comes in two flavours – online and offline. The protocol can be easily protected against online dictionary attacks by blocking the user access after some unsuccessful tries. On the other hand the off-line dictionary attacks can (and should) be prevented by the PAKE protocol itself.

*Related work.* The research on PAKE protocols started with EKE (Encrypted key exchange) protocol based on Diffie-Hellman key exchange. EKE was proposed by Bellowin and Merritt in [1], however, the paper provides only very informal proof of security. This original work spawned a lot of new research ideas.

Observing recent work, Bellare, Pointcheval and Rogaway conclude that although many new PAKE protocols are proposed, the theory is lagging behind. They therefore define a security model for PAKE protocols and prove the correctness of EKE. Boyko, MacKenzie and Patel [14] proposed 2PAKE protocols called PAK and PAK-X. They defined a new security model based on the model of Shoup [27]. Security of PAK is proved in the random oracle model under decisional Diffie-Hellman assumption. PAK is extended to a protocol PAK-X. It is built on the idea of a server which owns a user password verifier and the client stores a plaintext password. The authors formally proved the security of PAK-X, even when the server is compromised.

The work of MacKenzie [16] is based on [14], he introduced a protocol PAK-Z. Compared to PAK, PAK-Z has public and secret key for signature scheme. Here the client does a non-interactive proof of knowledge of his secret key.

The other research direction was pursued by Bresson, Chevassut and Pointcheval [13]. They introduced PAKE protocol OEKE (One encryption key exchange), whose advantage is in efficiency, because only one flow of sended data is encrypted.

Kwon, Jeong, Sakurai and Lee [7] deal with a multi-party scenario with a trusted server where each participant owns a different password. The goal of their protocols $\text{PAMKE}_1$ and $\text{PAMKE}_2$ is a group authentication and they note that designing PAKE protocols with trusted but curious server is quite involved task. Trusted server means that the server performs protocol steps and do not manipulate data in a different way. Curious means, that the server is honest, but we do not want it to know the computed session key. Another group authentication protocol was proposed by Lee, Hwang and Lee in [2]. The LHL protocol is however not secure as showed by Abdalla, Bresson and Chevassut

in [3] where they propose a new protocol secure against this attack. Choo [4] suggested another attack on LHL protocol.

*Our contribution.* We were inspired by the LHL protocol [2]. However in [3, 4] it is shown that this protocol is not secure. We propose a new PGAKE protocol based on the LHL and prove that this protocol is secure in a random oracle model and ideal cipher model under decisional Diffie-Hellmann assumption. The security model is adopted from [1, 5–7]. Our construction is secure against the attacks from [3, 4]. Second, every participant has his own secret password (compared with the protocol suggested in [3]) and because of this, there are not problems with adding a new participant and with compromising some participant. Our main contribution is proof of security (denoted as AKE-fs, see Definition 11) of our protocol.

## 2 Preliminaries

In this section, we establish the most important notation. If you are familiar with the standard notation in cryptography, it should be safe to skip this section.

### 2.1 Basic definitions

*Random choice* of an element $M$ from a finite set $S$ where the element $M$ is chosen uniformly is denoted as $M \xleftarrow{\$} S$. By $M_1 \parallel M_2$ we denote *concatenation* of two strings $M_1$ and $M_2$.

Random oracle is a function $f : M \to Y$ uniformly chosen from the set $Func(M, Y)$ of all functions with domain $M$ and range $Y$. Let us define $Z_n^* = \{x | 1 \leq x < n \wedge$ the greatest common divisor of integers $x, n$ is $1\}$.

We say that Turing machine $A$ has oracle access to Turing machine $B$ if machine $A$ can use $B$ as a function. We denote this fact as $A^B$. Symbol $\perp$ represents undefined value.

A symmetric encryption scheme $E = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ is defined as $\mathcal{E} : P \times K \to C$, $\mathcal{D} : C \times K \to P$, where $P$, $K$ and $C$ are sets of plaintexts, keys and ciphertexts, $\mathcal{G}$ is key-generation algorithm, $\mathcal{E}$ is encryption algorithm and $\mathcal{D}$ is decryption algorithm. For all $p \in P$ and all $k \in K$ it holds that $\mathcal{D}(\mathcal{E}(p, k), k) = p$. We say that encryption scheme with set of keys $K$ is ideal encryption scheme [9] (or is modeled in ideal cipher model) if is equivalent to family of $2^{|K|}$ independent random permutations. Message authentication code scheme (MAC) is denoted as $M = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrf})$, where Gen is key-generation algorithm, MAC is tag-generation algorithm and Vrf is verification algorithm. Mac computes a tag $\tau = \mathrm{Mac}_k(m)$ for message $m$ with use of key $k$. Vrf verifies a pair message, tag $(m, \tau)$ with use of key $k$, it returns 1 ($\mathrm{Vrf}_k(m, \tau) \to 1$) if the tag is valid for corresponding message $m$ and key $k$, 0 otherwise. A MAC scheme is existentially unforgeable under an adaptive chosen-message attack [10], if the adversary is not able to forge a valid tag on any message he has not been asked by his Mac oracle. We denote this property MAC-forge.

### 2.2 Protocols and adversaries

A single execution of a protocol is called a session. The set of protocol participants is $\mathcal{C} \cup \mathcal{S}$, where $\mathcal{C} = \{P_1, P_2, \dots P_n\}$ is set of clients and $\mathcal{S}$ is set of servers. For simplicity, we assume that $|\mathcal{S}| = 1$. Each client $P_i \in \mathcal{C}$ has a password $pw_i$ called long-lived key (LL-key) and server $S$ has a vector of clients passwords $\langle pw_{S,P_i} \rangle_{P_i \in \mathcal{C}}$ ($pw_i = pw_{S,P_i}$ for all $P_i \in \mathcal{C}$ in symmetric case, otherwise they are different in asymmetric case). The $j$-th instance of participant $P_i$ is denoted as $\Pi_i^j$ and $ID(P_i)$ is a unique identifier of participant $P_i$ (analogously $j$-th instance of server $S$ is denoted as $\Psi^j$). A group of participants $P_{i_1}, P_{i_2}, \dots, P_{i_k}$ is denoted as $Grp_{i_1,i_2,\dots,i_k}$.

**Definition 1.** [8] A protocol is a triple $\mathcal{P} = (\Pi, \Psi, LL)$, where $\Pi$ specifies how each client behaves, $\Psi$ specifies how server behaves and $LL$ specifies the distribution of long-lived keys.

A function is said to be negligible, if it decreases faster than inverse of any polynomial function.

**Definition 2.** A function $f : N \to R^+$ is negligible, if for every constant $c > 0$ there exists an integer $n_0 \in N$ such that $f(n) < \frac{1}{n^c}$ holds for all $n \in N, n > n_0$.

**Definition 3.** An adversary is a probabilistic polynomial-time Turing machine with oracle access to several other Turing machines. Running time of an adversary $\mathcal{A}$ is the length of description of $\mathcal{A}$ plus the worst case running time of $\mathcal{A}$.

Let $\mathcal{C}$ be a cryptographic construction (algorithm), $\mathcal{A}$ be an adversary and xxx be any problem on $\mathcal{C}$ (such as collision resistance of hash function, or discrete logarithm in a group $G$). $\mathbf{Adv}^{\text{xxx}}_{\mathcal{C},\mathcal{A}}$ is a measure of adversary's advantage defined as a probability, that $\mathcal{A}$ succeeds to solve the problem xxx for $\mathcal{C}$. Sometimes, the advantage depends on some parameter, such as time of execution, length of the algorithm's input or the number of some queries. Let $a_1, a_2, \dots a_n$ be parameters needed for the security definition, then the adversary's advantage is denoted as $\mathbf{Adv}^{\text{xxx}}_{\mathcal{C},\mathcal{A}}(a_1, a_2, \cdots a_n)$.

In this paper we adopt a Dolev-Yao model of an adversary, where the adversary intercepts whole communication during the execution of the protocol. The adversary can delay, change or deliver messages out of order, start a new execution of protocol, acquire LL-key of some participants and acquire given session key. All abilities of the adversary are modelled through oracles defined in Section 4.

We use the notion of Decisional Diffie-Hellman assumption in our security proofs.

**Definition 4** (Decisional Diffie-Hellmann assumption – DDH)**.** Let $G$ be a cyclic group of order $q$ with generator $g$ and $\mathcal{D}$ be an adversary. Two distributions are defined:

$DDH^* = \{(g^x, g^y, g^{xy}) | x, y \xleftarrow{\$} Z_q^*\}$ and
$DDH^{\$} = \{(g^x, g^y, g^w) | x, y, w \xleftarrow{\$} Z_q^*\}$. The DDH problem for input $(u, v, w)$ is to distinguish, from which distribution is it. The DDH assumption holds in a cyclic group $G$ if and only if the advantage of every adversary $\mathcal{D}$ on DDH problem in time $T$ is negligible. This advantage is denoted as $\mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{D}}(T)$ and computed as:

$$\mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{D}}(T) = |\Pr[\mathcal{D}(DDH^*) \to 1] - \Pr[\mathcal{D}(DDH^{\$}) \to 1]|.$$

Abdalla et al. [3] defined Parallel Decisional Diffie-Hellman assumption and challenger $Chall^{\beta}(\cdot)$.

**Definition 5** (Parallel Decisional Diffie-Hellmann assumption – $\mathrm{PDDH}_n$)**.** Let $G$ be a cyclic group of order $q$ with generator $g$ and $\mathcal{D}$ be an adversary. Two distributions are defined:
$PDH_n^* = \{(g^{x_1}, g^{x_2}, \ldots, g^{x_n}, g^{x_1 x_2}, g^{x_2 x_3}, \ldots, g^{x_n x_1}) | x_1, x_2, \ldots, x_n \xleftarrow{\$} Z_q^*\}$ and
$PDH_n^{\$} = \{(g^{x_1}, g^{x_2}, \ldots, g^{x_n}, g^{y_1}, g^{y_2}, \ldots, g^{y_n}) | x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n \xleftarrow{\$} Z_q^*\}$, where $n > 2$. The $\mathrm{PDDH}_n$ problem for input $(u_1, u_2, \ldots, u_n, w_1, w_2, \ldots, w_n)$ is to distinguish, from which distribution is it. The $\mathrm{PDDH}_n$ assumption holds in a cyclic group $G$ if and only if the advantage of every $\mathcal{D}$ on $\mathrm{PDDH}_n$ problem in time $T$ is negligible. This advantage is denoted as $\mathbf{Adv}^{\mathrm{PDDH}_n}_{G,\mathcal{D}}(T)$ and computed as:

$$\mathbf{Adv}^{\mathrm{PDDH}_n}_{G,\mathcal{D}}(T) = |\Pr[\mathcal{D}(PDH_n^*) \to 1] - \Pr[\mathcal{D}(PDH_n^{\$}) \to 1]|.$$

In [3], it was proved that for a group $G$, time $T$, integer $n > 2$ and distinguisher $\mathcal{D}$ the $\mathrm{PDDH}_n$ and DDH problems are equivalent in $G$:

$$\mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{D}}(T) \leq \mathbf{Adv}^{\mathrm{PDDH}_n}_{G,\mathcal{D}}(T) \leq n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{D}}(T)$$

$Chall^{\beta}(S)$ is an algorithm that on input $S$ outputs vectors from the distribution $PDH_n^*$, if the bit $\beta = 0$, otherwise it outputs vectors from the distribution $PDH_n^{\$}$. If the same $S$ is given on input again, then the same vectors are returned.

## 3 LHL protocol and its shortcomings

In this section, we review Lee, Hwang, Lee (LHL) protocol [2] and two attacks on this protocol. The goal of LHL protocol is to authenticate a group $P_1, \ldots, P_n$ of participants which share a common secret password $pw \in \mathcal{DICT}$ and establish a group session key.

The protocol works in a cyclic group $G$ of order $q$ ($q$ is a prime) with a generator $g$. We assume that participants share an *ideal* symmetric encryption scheme $E = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ where $\mathcal{E} : G \times \mathcal{DICT} \to G$, $\mathcal{D} : G \times \mathcal{DICT} \to G$. Moreover, let $\mathcal{H}$ and $\mathcal{H}'$ be two pseudorandom hash functions, such that $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{l_{\mathcal{H}}}$ and $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^{l_{\mathcal{H}'}}$.

Supposing that participants $P_1, \ldots, P_n$ are arranged along a circle, the protocol works in the following steps:

1. Establish a temporary key $K_i$ between each pair of neighbours $P_i, P_{i+1}$ using the encrypted Diffie-Hellman key exchange:
   (a) Each participant $P_i$ selects a random number $x_i \xleftarrow{\$} Z_q^*$;
   (b) $P_i$ sends $ID(P_i) \parallel \mathcal{E}_{pw}(g^{x_i})$ to his two neighbours;[1]
   (c) Upon receiving and decrypting $g^{x_{i\pm1}}$, participant $P_i$ computes $K_i = \mathcal{H}((g^{x_{i+1}})^{x_i})$ and $K_{i-1} = \mathcal{H}((g^{x_{i-1}})^{x_i})$.
2. Participant $P_i$ computes value $w_i = K_{i-1} \oplus K_i$ (a xor of temporary keys with his neighbours) and broadcasts $ID(P_i) \parallel w_i$.
3. Session key $sk = \mathcal{H}'(K_1 \parallel K_2 \parallel \ldots \parallel K_n)$ is established by all participants using following procedure:
   (a) Participant $P_i$ can reconstruct $K_{i+1}$ as $w_{i+1} \oplus K_i$;
   (b) Having established $K_{i+1}$, he can continue in similar manner and reconstruct $K_{i+2}, K_{i+3}, \ldots, K_n, K_1, \ldots K_{i-2}$
   (c) $P_i$ can compute $sk = \mathcal{H}'(K_1 \parallel K_2 \parallel \ldots \parallel K_n)$.

### 3.1 Attacking LHL

In the introduction we have already mentioned that PAKE protocols are susceptible to dictionary attacks. Even if one guarantees that the distribution of session keys generated by the protocol is the same for every password, the attacker might use correlations between different session keys to quickly pinpoint the user password. Simply put, the protocol is secure if established session keys between different sessions are uncorrelated, *i.e.* revealing previous session keys does not bring any advantage to the attacker (adversary). Another important part of the security is a *forward secrecy* which models misuse of user long-lived key (password). Informally, revealing user passwords should not bring any advantage to the adversary trying to break *previous* session keys.

Now, we will show the two different attacks on LHL protocol. The first attack was described in [3] and goes as follows: First, the attacker convinces the participant $P_1$ to start the group authentication twice with another three participants. Denote values chosen by $P_1$ in both instances as $x_1$ and $x_1'$ and let $z_1$ and $z_1'$ be corresponding encrypted values that are sent. Attacker now impersonates participants $P_2$, $P_3$ and $P_4$ in both sessions as follows: $P_2 \to P_1 : (ID(P_2)||z_1')$, $P_3 \to P_1 : (ID(P_3)||z_1)$, $P_4 \to P_1 : (ID(P_4)||z_1')$ and similarly for the other session. After this exchange, all Diffie-Hellman exchanges will result in $g^{x_1 x_1'}$ and therefore $w_i = w_i' = 0$ for all participants. Thus, the attacker can simply send the second round of messages and finish the protocol.

In essence, this attack forces participant $P_1$ to establish the same group session key twice (on top of believing to be communicating with $P_2$, $P_3$ and $P_4$). Although the attacker will not obtain the session key itself, compromising one instance of participant $P_1$ will break the other one, thus violating our security requirements.

[1] Original LHL protocol broadcasts this information. From our point of view it is redundant

Next we present an *unknown key share* attack on LHL described in [4]. In this attack, the adversary tricks participant $P_1$ to establish a key with participants $P_2, P_3$ while $P_2$ and $P_3$ believe that the key is shared with $P_4 \neq P_1$. The attack starts with a group of three participants $P_1, P_2, P_3$. When each participant $P_i$ sends $(ID(P_i)||z_i)$, the adversary intercepts $(ID(P_1)||z_1)$ and sends $(ID(P_4)||z_1)$ instead. Similarly, the adversary intercepts $(ID(P_1)||w_1)$ and sends $(ID(P_4)||w_4)$. Now, the participant $P_1$ established a key with $P_2$ and $P_3$ but both of them believe that they share the key with $P_4$ instead.

## 4 Security model

In this section we present a model based on [1, 12], later extended in [6] and adapted for group key exchange in [7]. For identification of a concrete session and an instance of a partner in the session we defined notions *session identifier* and *partnering*.

**Definition 6.** Session identifier (sid) is a unique identifier of the session. It is the same for all participants in the session. Session identifier of instance $\Pi_i^j$ is denoted as $sid_i^j$. For a server instance $\Psi^s$ session identifier is denoted as $sid^s$.

If instances $\Pi_i^j$, $\Pi_k^l$ and $\Psi^s$ are in the same session, then $sid_i^j = sid_k^l = sid^s$.

**Definition 7.** Partner identifier $pid_i^j$ for instance $\Pi_i^j$ is set of all identifiers of instances with whom $\Pi_i^j$ wants to establish a session key. Instances $\Pi_i^j$ and $\Pi_k^l$ are partners, if

- $sid_i^j = sid_k^l \neq \perp$
- $\Pi_i^j \in pid_k^l$ and $\Pi_k^l \in pid_i^j$

An adversary controls whole communication. He can stop sended message, send message $M$, deliver messages out of order and intercept communication. His abilities are modelled using the following oracles:

- Send$(\Pi_i^j, M)$ – sends the message $M$ to the instance $\Pi_i^j$ in session $sid_i^j$ and returns a reply of $\Pi_i^j$ (according to the execution of the protocol). This oracle query simulates an active attack of the adversary.
- Send$(\Psi^s, M)$ – similarly to the $Send(\Pi_i^j, M)$. This oracle query sends the message $M$ to the instance of server $\Psi^s$ in the session $sid^s$ and returns the reply of $\Psi^s$.
- Execute$(Grp_{i_1,i_2,\ldots,i_k}, S)$ – this oracle starts execution of the protocol between participants $P_{i_1}, P_{i_2}, \ldots, P_{i_k}$ and server $S$. The result is a full copy of messages sent during execution of the protocol. This query models a passive attack, where adversary eavesdrops the execution of the protocol.
- Reveal$(\Pi_i^j)$ – if the instance $\Pi_i^j$ has established session key $sk$, then the oracle returns $sk$ else return $\perp$. This oracle models scenario of session key leakage.

– Corrupt($P_i$) – this query returns the LL-key $pw_i$ of participant $P_i$. This oracle models forward secrecy. (Such definition of Corrupt query is in a weak corruption model. In a strong corruption model Corrupt($P_i$) returns an internal state of all instances of participant $P_i$ too.)
– Test($\Pi_i^j$) – This query can be used only on a fresh/fs-fresh instance (see Def. 8). First a random bit $b \xleftarrow{\$} \{0,1\}$ is chosen. If instance $\Pi_i^j$ has not established session key $sk$, then $\perp$ is returned. If $b = 0$, then the real session key $sk$ is returned else (if $b = 1$) random string $sk' \xleftarrow{\$} \{0,1\}^{|sk|}$ is returned.

**Definition 8** (Fresh and fs-fresh instance)**.** Instance $\Pi_i^j$ is *fresh*,

1. if oracle query Reveal was not made on the instance $\Pi_i^j$ and its partners,
2. and if Corrupt query was not made on any protocol's participant in any session.

Instance $\Pi_i^j$ is *fs-fresh*,

1. if oracle query Reveal was not made on the instance $\Pi_i^j$ and its partners,
2. and if Corrupt query was not made on any protocol's participant in any session before Test query or Send query was not made on instance $\Pi_i^j$.

Forward secrecy is security feature of a protocol and it is defined by Corrupt queries on the protocol. Informally, the protocol has forward secrecy property, if and only if revealing of LL-keys does not compromise previous established session keys.

**Definition 9.** Advantage $\mathbf{Adv}^{\mathrm{AKE}}_{\mathcal{P},\mathcal{A}}(k)$ of an adversary $\mathcal{A}$ attacking a protocol $\mathcal{P}$ in aforementioned model without forward secrecy with security parameter $k$ is defined by a following game:
GameAKE$_{\mathcal{P},\mathcal{A}}$:

– $\mathcal{A}$ can ask queries to Send, Reveal and Execute oracles multiple times.
– Test query can $\mathcal{A}$ ask only once and on a fresh instance.
– $\mathcal{A}$ returns bit $b'$.

Let $Succ$ denote the event, that $b = b'$, where $b$ is the bit randomly chosen during Test oracle. Then $\mathbf{Adv}^{\mathrm{AKE}}_{\mathcal{P},\mathcal{A}}(k) = |2 \cdot \Pr[Succ] - 1|$.

**Definition 10.** Advantage $\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathcal{P},\mathcal{A}}(k)$ of an adversary $\mathcal{A}$ attacking a protocol $\mathcal{P}$ in aforementioned model with forward secrecy and security parameter $k$ is defined as follows:
GameAKE-fs$_{\mathcal{P},\mathcal{A}}$:

– $\mathcal{A}$ can ask queries to Send, Reveal, Execute and Corrupt oracles multiple times.
– Test query can $\mathcal{A}$ ask only once and on a fs-fresh instance.
– $\mathcal{A}$ returns a bit $b'$.

Let $Succ$ denote the event, that $b = b'$, where $b$ is the bit randomly chosen during Test oracle. Then $\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathcal{P},\mathcal{A}}(k) = |2 \cdot \Pr[Succ] - 1|$.

**Definition 11.** We say a protocol $\mathcal{P}$ is AKE (AKE-fs) secure multi-party PAKE protocol without (with) forward secrecy, if for all adversaries $\mathcal{A}$ running in polynomial time holds:

- all participant instances which are partners have the same session key,
- $\mathbf{Adv}^{\mathrm{AKE}(-\mathrm{fs})}_{\mathcal{P},\mathcal{A}}(k) \leq \frac{Q(k)}{|\mathcal{DICT}|} + \varepsilon(k)$, where $\varepsilon(k)$ is negligible and $Q(k)$ denotes the number of on-line attacks (all Send queries to clients, server $S$ and all Corrupt queries). $\mathcal{DICT}$ is a set of all possible passwords.

## 5 Our protocol

Our design goals for the new protocol are following:

- Enable group-based authentication with a *distinct password per user*. This however requires a presence of a *trusted* server.
- Protect against the previously mentioned attacks.

We meet both these design goals by replacing the first step of LHL protocol with a secure communication through the trusted server. Because of this secure communication, the attacker can no longer exchange user identities by switching messages.

Similarly to the LHL, our protocol works with a cyclic group of order $q$ ($q$ is a prime) with generator $g$. Again, we will use two pseudorandom hash functions $\mathcal{H}$ and $\mathcal{H}'$ ($\mathcal{H} : \{0,1\}^* \to \{0,1\}^{l_{\mathcal{H}}}$ and $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^{l_{\mathcal{H}'}}$). What is new is the presence of a trusted server. Every participant $P_i$ has password $pw_i \in \mathcal{DICT}$, which is shared with the server. To establish a secure connection to the server, we use arbitrary secure 2PAKE protocol denoted as 2P with the length of the session key $l_k$. We assume a symmetric encryption scheme modeled as an ideal cipher $E = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ defined as $\mathcal{E} : G \times \{0,1\}^{l_k} \to G$, $\mathcal{D} : G \times \{0,1\}^{l_k} \to G$ and an existentially unforgeable under an adaptive chosen-message attack secure message authentication scheme $M = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrf})$.

**Protocol MLHL (Multi-LHL):**

1. Each participant $P_i$ establishes a key $sk_i$ with the server $S$ using 2P protocol.
2. Establish a temporary key $K_i$ between each pair of neighbours:
   (a) Each participant $P_i$ chooses a random $x_i$, computes $z_i = g^{x_i}$ and sends to the server message $P_i \to S : ID(P_i)||z_i^* = \mathcal{E}_{sk_i}(z_i)$.
   (b) Server decrypts $z_i^*$ and sends following messages to the participants $P_{i-1}$ and $P_{i+1}$:
   $S \to P_{i-1} : ID(S)||ID(P_i)||\mathcal{E}_{sk_{i-1}}(z_i)$
   $S \to P_{i+1} : ID(S)||ID(P_i)||\mathcal{E}_{sk_{i+1}}(z_i)$
   (c) Each $P_i$ decrypts received messages to obtain values $z_{i-1}$ and $z_{i+1}$ and computes $K_i = \mathcal{H}(z_{i+1}^{x_i})$, $K_{i-1} = \mathcal{H}(z_{i-1}^{x_i})$.

3. Each participant $P_i$ computes $w_i = K_{i-1} \oplus K_i$, then he computes MAC $\tau_i = \mathrm{Mac}_{K_i}(ID(P_i)||w_i)$ and broadcasts message $(ID(P_i)||w_i||\tau_i)$.
4. When $P_i$ receives messages $(ID(P_j)||w_j||\tau_j)$ from all other participants, he computes $K_j = \mathcal{H}(g^{x_{j-1}x_j})$ for all $j \in \{1, ..., n\}$ using the values $w_j$ and $K_{i-1}$, in direction to the left (from $K_{i-1}, \ldots, K_n, \ldots, K_{i+1}, K_i$). During this computation, he verifies for received values $ID(P_j)$ and $w_j$ their tags $\tau_j$. For example, he starts with computing $K'_{i-2} = w_{i-1} \oplus K_{i-1}$, $\mathrm{Vrf}_{K_{i-1}}(ID(P_{i-1})||$ $w_{i-1}, \tau_{i-1})$ and ends with $K'_i = w_{i+1} \oplus K_{i+1}$, $\mathrm{Vrf}_{K_{i+1}}(ID(P_{i+1})||w_{i+1}, \tau_{i+1})$. If all tag values are correct, then $P_i$ continues with the next step, otherwise terminates.
5. $P_i$ computes the session key $sk = \mathcal{H}'(K_1 \| K_2 \| \ldots \| K_n)$.

The verification phase disables the adversary to change sent messages in the way, that participants do not know about this change.

### 5.1 Security of MLHL protocol

Let $G$ be a cyclic group with generator $g$, for which the DDH assumption holds. Let $\mathcal{H}$ and $\mathcal{H}'$ be modeled as random oracles, where $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{l_{\mathcal{H}}}$ and $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^{l_{\mathcal{H}'}}$. Let 2P be an arbitrary secure 2PAKE protocol with length of the session key $l_k$, let $E = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ be symmetric encryption scheme defined as $\mathcal{E} : G \times \{0,1\}^{l_k} \to G$, $\mathcal{D} : G \times \{0,1\}^{l_k} \to G$ and modeled as an ideal cipher. Let $M = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrf})$ be an existentially unforgeable under adaptive chosen-message attack secure message authentication scheme. Symbol $\varepsilon$ denotes a negligible function, $q_{\mathcal{E}}$ number of encryption queries, $q_{\mathcal{D}}$ number of decryption queries, $q_{send}, q_{execute}, q_{reveal}$ is number of Send, Execute, Reveal queries the attacker makes in underlying 2P protocol during GameAKE-$\mathrm{fs}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}$. Polynomial $p(\cdot)$ denotes the number of instances of the protocol MLHL executed through the Execute oracle or through the sequence of Send queries. Symbol $\mathcal{A}_X$ denotes adversary attacking construction $X$ on its security property. Running times of adversaries $\mathcal{A}_{\mathrm{MLHL}}$, $\mathcal{A}_{\mathrm{2P}}$, $\mathcal{A}_M$ and $\mathcal{A}_{\mathrm{DDH}}$ are denoted $T, t, t', t''$ and $k$ is security parameter.

**Theorem 1.** *Assume that every participant $P_i$ has a secret key $pw_i \in \mathcal{DICT}$, which is shared with the server $S$. We suppose, that the adversary $\mathcal{A}_{\mathrm{MLHL}}$ establishes $p(k)$ sessions during GameAke$_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}$ between $n$ participants for some polynomial $p(\cdot)$. Then the advantage of the adversary $\mathcal{A}_{\mathrm{MLHL}}$ in attacking protocol* MLHL *is*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}(k,T) \leq 2\Bigg( & \frac{3(q_{\mathcal{E}}+q_{\mathcal{D}})^2}{2|G|} + \frac{3p(k)\cdot n\cdot q_{\mathcal{D}}}{2^{l_k}} + 4\mathbf{Adv}^{\mathrm{MAC-forge}}_{M,\mathcal{A}_M}(t') \\ & + p(k)\cdot n\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{2P},\mathcal{A}_{\mathrm{2P}}}(t, q_{execute}, q_{send}, q_{reveal}) + 2\varepsilon \\ & + \frac{np(k)^2}{2^{l_k+1}} + 5p(k)\cdot n\cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'') + 8q_{\mathcal{E}}/2^{l_k}\Bigg).\end{aligned}$$

Looking at the definition of fs-fresh instance on which adversary makes a Test query we have following cases of Corrupt query usage (on instance in Test query) during the game GameAKE-$\text{fs}_{\text{MLHL},\mathcal{A}_{\text{MLHL}}}$:

- $Case_1$: No Corrupt query was made during the execution of the game GameAKE-$\text{fs}_{\text{MLHL},\mathcal{A}_{\text{MLHL}}}$. In this case the adversary can ask Send, Execute and Reveal queries.
- $Case_2$: In this case, there must be at least one Corrupt query and all Corrupt queries were made after a Test query in the game GameAKE-$\text{fs}_{\text{MLHL},\mathcal{A}_{\text{MLHL}}}$ (note that in this case the session key was established for instance on which Test query was made). Here Send, Execute and Reveal queries are allowed.
- $Case_3$: In this case, there must be at least one Corrupt query and some Corrupt query was made before a Test query in the game GameAKE-$\text{fs}_{\text{MLHL},\mathcal{A}_{\text{MLHL}}}$. In this case only Execute and Reveal queries are allowed, due to preservation of fs-fresh property (adversary can not ask Send query on instances of other participants in the same session, because if he starts to ask Send queries in the session, he must ask Send queries on instance on which he will ask a Test query to finish the protocol execution correctly).

Therefore, we can divide advantage of the adversary attacking on AKE-fs security into advantage of the adversary in every of these cases:

$$\begin{aligned}\mathbf{Adv}^{\text{AKE}-\text{fs}}_{\text{MLHL},\mathcal{A}_{\text{MLHL}}}(T) = \mathbf{Adv}^{\text{AKE}-\text{fs}}_{\text{MLHL},\mathcal{A}_{\text{MLHL}},Case_1}(T) + \mathbf{Adv}^{\text{AKE}-\text{fs}}_{\text{MLHL},\mathcal{A}_{\text{MLHL}},Case_2}(T) \\ + \mathbf{Adv}^{\text{AKE}-\text{fs}}_{\text{MLHL},\mathcal{A}_{\text{MLHL}},Case_3}(T).\end{aligned}$$

We prove the theorem for every case in three lemmas by sequence of games, starting with the game $G_0$ simulating the real protocol. In these games we simulate participants of the protocol and their behavior. By $Succ_i$ we denote that $b = b'$ in the game $G_i$, where $b$ was randomly chosen bit in Test query and $b'$ is the output of the adversary.
For simplicity we suppose, that the adversary asks Execute queries on group with the number of users $n$. Similarly when the protocol is simulated through *Send* queries, we assume that the number of users is $n$ too.

*Proof.* The proof of theorem is given in Appendix A.

**Security of the MLHL protocol against attacks on LHL protocol:** The first attack is not possible, because in every session participant $P_i$ establishes key $sk_i$ with server $S$ with use of protocol 2P, therefore the keys are different in every session and with high probability are not repeated. If adversary repeats some messages, they will be decrypted and encrypted with different passwords and if the adversary can not manipulate the messages that they go through verification step (DDH problem and MAC property), protocol terminates.
The second type of attack is not possible due to MAC property too. Adversary can manipulate messages that participants $P_2$ and $P_3$ think, that they receive messages from $P_4$ (adversary overrides value $P_1$ by value $P_4$ in messages sent by server), but he does not know to compute correct $\tau_4$. Therefore the execution of protocol terminates during verification step.

## 6 Conclusion

We have proposed and proved security of the MLHL protocol, which is secure PGAKE protocol for participants with different passwords. This protocol is based on the LHL protocol, but his advantage is in security against mentioned attacks and in usage of different passwords per user.

**Acknowledgement.** This paper was supported by VEGA grant number 1/0259/13 and by Comenius University grant number UK/407/2013.

## References

1. Steven M. Bellovin and Michael Merritt,Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks, In IEEE Computer Society Symposium on Research in Security and Privacy, pp. 72-84, IEEE Computer Society Press, 1992.
2. Lee, Su-Mi and Hwang, Jung Yeon and Lee, Dong Hoon, Efficient Password-Based Group Key Exchange, Trust and Privacy in Digital Business, First International Conference, TrustBus'04, pp. 191-199, LNCS 3184, Springer, 2004.
3. Michel Abdalla and Emmanuel Bresson and Olivier Chevassut, Password-based Group Key Exchange in a Constant Number of Rounds, Public Key Cryptography - PKC'06 - 9th International Conference on Practice and Theory in Public Key Cryptography, pp. 427–442, LNCS 3958, Springer, 2006.
4. Choo, Kim-Kwang Raymond, On the Security Analysis of Lee, Hwang & Lee (2004) and Song & Kim (2000) Key Exchange / Agreement Protocols, Informatica, 17, pp. 467-480, IOS Press, 2006.
5. Michel Abdalla and Pierre-Alain Fouque and David Pointcheval, Password-based authenticated key exchange in the three-party setting, PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography, pp. 65–84, LNCS 3386, Springer, 2005.
6. Mihir Bellare and David Pointcheval and Phillip Rogaway, Authenticated Key Exchange Secure Against Dictionary Attacks, Advances in Cryptology – EUROCRYPT'00, International Conference on the Theory and Application of Cryptographic Techniques, pp 139–155, LNCS 1807, Springer, 2000.
7. Jeong Ok Kwon and Ik Rae Jeong and Kouichi Sakurai and Dong Hoon Lee, Password-authenticated multiparty key exchange with different passwords, IACR Cryptology ePrint Archive, 2006.
8. Mihir Bellare and Phillip Rogaway, Provably secure session key distribution: The Three Party Case, Proceedings of the twenty-seventh annual ACM symposium on Theory of computing, STOC '95, pp. 57–66, ACM, 1995.
9. Jean-Sébastien Coron and Jacques Patarin and Yannick Seurin, The Random Oracle Model and the Ideal Cipher Model Are Equivalent, Advances in Cryptology - CRYPTO'08, 28th Annual International, pp. 1–20, LNCS 5157, Springer, 2008.
10. Katz, Jonathan and Lindell, Yehuda, Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series),Chapman & Hall/CRC, 2007.
11. Jonathan Katz and Rafail Ostrovsky and Moti Yung, Forward Secrecy in Password-Only Key Exchange Protocols, Security in Communication Networks, Third International Conference, pp. 29–44, LNCS 2576, Springer, 2003.

12. Mihir Bellare and Ran Canetti and Hugo Krawczyk, A modular approach to the design and analysis of authentication and key exchange protocols, STOC'98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 419–428, ACM, 1998.
13. E. Bresson and O. Chevassut and D. Pointcheval, Security proofs for an efficient password-based key exchange, Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS'03, pp. 241–250, ACM, 2003.
14. Victor Boyko and Philip Mackenzie and Sarvar Patel, Provably secure password-authenticated key exchange using Diffie-Hellman, Advances in Cryptology - EUROCRYPT'00, International Conference, pp. 156–171, LNCS, Springer, 2000.
15. David P. Jablon, Strong Password-Only Authenticated Key Exchange, SIGCOMM Computer Communication Review 26, pp. 5–26, ACM, 1996.
16. Philip MacKenzie, The PAK suite: Protocols for Password-Authenticated Key Exchange, IEEE P1363.2, 2002.
17. Thomas Wu, The secure remote password protocol, Proceedings of the Network and Distributed System Security Symposium, NDSS'98, The Internet Society, pp. 97–111, 1998.
18. Oded Goldreich and Yehuda Lindell, Session-Key Generation using Human Passwords Only, Advances in Cryptology - CRYPTO'01, 21st Annual International Cryptology Conference, pp. 408–432, LNCS 2139, Springer, 2001.
19. Jonathan Katz and Rafail Ostrovsky and Moti Yung, Efficient Password-Authenticated Key Exchange using Human-Memorable Passwords, Advances in Cryptology - EUROCRYPT'01, International Conference on the Theory and Application of Cryptographic Techniques, pp. 475–494, LNCS, Springer, 2001.

# A Proof of Theorem 1

## A.1 Advantage of adversary in $Case_1$:

In this section we prove the AKE-fs security of the MLHL protocol in $Case_1$, where the adversary does not make any Corrupt queries.

**Lemma 1.** *The advantage of the adversary from $Case_1$ is:*

$$\begin{aligned}\boldsymbol{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}},Case_1}(T) \leq 2\Bigg( & \frac{(q_{\mathcal{E}}+q_{\mathcal{D}})^2}{2|G|} + \frac{p(k)\cdot n\cdot q_{\mathcal{D}}}{2^{l_k}} + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{M}(t') \\ & + p(k)\cdot n\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{2P}}(t, q_{execute}, q_{send}, q_{reveal}) \\ & + \frac{np(k)^2}{2^{l_k+1}} + 2p(k)\cdot n\cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'') + 4q_{\mathcal{E}}/2^{l-k}\Bigg).\end{aligned}$$

If no Corrupt queries are made, it is sufficient to prove AKE security instead of AKE-fs, therefore $\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}},Case_1}(T) = \mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}(T)$.

*Proof.* We start with the simulation of the real protocol.
**Game** $G_0$:
This is a game simulating the real protocol. From the definition 9 we have:

$$\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}(T) = 2\Pr[Succ_0] - 1.$$

Because 2P could represent arbitrary secure 2PAKE protocol, without the loss of generality we suppose that in the protocol 2P the communication is started by participant $P_i$, which sends the first message to the server. Moreover, we assume that the protocol has $l$ flows of messages and the last message ($l$-th) is sent by the server. By $sk_i = 2\mathrm{P}(P_i, S)$ we denote, that key $sk_i$ was computed with simulation of 2P between $P_i$ and $S$. When participant awaits more than one message, we denote it as a concatenation (see definitions of $\mathrm{Send}_3$ and $\mathrm{Send}_4$ oracles).
In this game we simulate Send and Execute oracles as described bellow (we skip the description of Test and Reveal queries, because they are straightforward from their definition). The simulation of Send queries is divided into $l + 4$ types of queries ($l$ is number of messages sent during 2P protocol). Such Send query represents concrete type of message, which was sent.

**Execute**$(\mathbf{P_1}, \ldots, \mathbf{P_n}, \mathbf{S})$
$msg_0^i = 2\mathrm{P}(P_i, S)$,
$x_1, \ldots, x_n \xleftarrow{\$} G$,
$\forall P_i, i \in \{1, \ldots, n\}$ compute $z_i = g^{x_i}$,
$z_i^* = \mathcal{E}_{sk_i}(g^{x_i})$,
$z_{i-1}^{**} = \mathcal{E}_{sk_{i-1}}(z_i)$, $z_{i+1}^{**} = \mathcal{E}_{sk_{i+1}}(z_i)$,
$msg_{1.1}^i = (ID(P_i)||\mathcal{E}_{sk_i}(z_i))$,
$msg_{1.2}^i = (ID(S)||ID(P_i)||z_{i-1}^{**})$,
$msg_{1.3}^i = (ID(S)||ID(P_i)||z_{i+1}^{**})$,
$\forall P_i, i \in \{1, \ldots, n\}$ compute $K_i = \mathcal{H}(g^{x_i x_{i+1}})$,
$w_i = K_{i-1} \oplus K_i$,
$\tau_i = \mathrm{Gen}_{K_i}(ID(P_i)||w_i)$
$msg_2^i = (ID(P_i)||w_i||\tau_i)$,
$sk = \mathcal{H}'(\mathcal{H}(K_1)\|...\|\mathcal{H}(K_n))$,
$msg_3^i =$ "accepted"
**return** $\{msg_0^i, msg_{1.1}^i, msg_{1.2}^i, msg_{1.3}^i, msg_2^i, msg_3^i\}_{i=1}^n$

$\text{Send}_1^1(\Pi_i^j, M)$

simulate first step of 2P protocol, message $M$ of the form $(ID(P_{i_1})||ID(P_{i_2})||\ldots||ID(P_{i_{n-1}})||ID(S))$ is sent to the instance $\Pi_i^j$ informing that the instance $\Pi_i^j$ is going to establish a session key with participants $P_{i_1}, P_{i_2}, \ldots, P_{i_{n-1}}$,

**return** the message, which is the result of simulation of the first step of 2P protocol.

$\vdots$

$\text{Send}_1^l(\psi^s, M)$

simulate the last step of 2P protocol,

**return** the last message of 2P protocol computed according to the rules of the 2P.

$\text{Send}_2^1(\Pi_i^j, M)$

$M$ is the last message sent by the server $\psi$ to $\Pi_i^j$ in 2P,

$sk_i = 2\text{P}(P_i, S)$,

$x_i \xleftarrow{\$} G$,

$z_i = g^{x_i}$, $z_i^* = \mathcal{E}_{sk_i}(z_i)$

**return** $(ID(P_i)||z_i^*)$

$\text{Send}_2^2(\psi^s, M)$

$M$ has the form $(ID(P_i)||M')$

$z_i = \mathcal{D}_{sk_i}(M)$,

$z_{i-1}^{**} = \mathcal{E}_{sk_{i-1}}(z_i)$,

$z_{i+1}^{**} = \mathcal{E}_{sk_{i+1}}(z_i)$

**return** $(ID(S)||ID(P_i)||z_{i-1}^{**})$, $(ID(S)||ID(P_i)||z_{i+1}^{**})$

$\text{Send}_3(\Pi_i^j, M_{i-1}||M_{i+1})$

$M_{i-1}$ and $M_{i+1}$ have the form $(ID(S)||ID(P_{i-1})||M'_{i-1})$ and $(ID(S)||ID(P_{i+1})||M'_{i+1})$

$z_{i-1} = \mathcal{D}_{sk_i}(M_{i-1})$, $z_{i+1} = \mathcal{D}_{sk_i}(M_{i+1})$,

$K_{i-1} = \mathcal{H}(z_{i-1}^{x_i})$, $K_i = \mathcal{H}(z_{i+1}^{x_i})$,

$w_i = K_{i-1} \oplus K_i$, $\tau_i = \text{Mac}_{K_i}(ID(P_i)||w_i)$

**return** $(ID(P_i)||w_i||\tau_i)$

$\text{Send}_4(\Pi_i^j, M_0||\ldots||M_{i-1}||M_{i+1}||\ldots||M_n)$

$M_j$ has the form $(ID(P_j)||w_j||\tau_j)$,

$j \in \{0, \ldots, i-1, i+1, \ldots, n\}$,

**if** $\text{Vrf}_{K_{i-1}}(ID(P_{i-1})||w_{i-1}, \tau_{i-1}) = 1$

**then** $K_{i-2} = w_{i-1} \oplus K_{i-1}$, ...

**if** $\text{Vrf}_{K_{i+1}}(ID(P_{i+1})||w_{i+1}\tau_{i+1}) = 1$

**then** $K_i = w_{i+1} \oplus K_{i+1}$,

$sk = \mathcal{H}'(\mathcal{H}(K_1)\|...\|\mathcal{H}(K_n))$,

**return** "accept"

**else if** any of MAC verifications fails, **return** "terminated"

**Game** $G_0'$:
In this game we simulate encryption and decryption oracles. We work with the list $\Lambda_{\mathcal{E}}$ of tuples $(type, sid_i^j, i, \alpha, sk, z, z^*)$, where we store previous answers of encryption/decryption queries. $Type$ takes values $enc/dec$, $sid_i^j$ is session ID of instance $\Pi_i^j$, $\alpha$ is value used in other games, $sk$ is encryption/decryption key and $z^* = \mathcal{E}_{sk}(z)$. Moreover, we use the list $\Lambda_{2\mathrm{P}}$ of tuples $(sid, i, sk)$ where we store previously established session keys $sk$ in 2P protocol in session $sid$ for participant $P_i$. We simulate encryption and decryption as follows:

- $\mathcal{E}_{sk}(z)$ – if $(\cdot, \cdot, \cdot, \cdot, sk, z, z^*) \in \Lambda_{\mathcal{E}}$, we return $z^*$ otherwise we choose $z^* \xleftarrow{\$} G$, if $(\cdot, \cdot, \cdot, \cdot, sk, \cdot, z^*) \in \Lambda_{\mathcal{E}}$, we stop the simulation and the adversary wins (because such situation represents collision). Otherwise we add a record $(enc, \perp, \perp, \perp, sk, z, z^*)$ to $\Lambda_{\mathcal{E}}$ and return $z^*$.
- $\mathcal{D}_{sk}(z^*)$ – if $(\cdot, \cdot, \cdot, \cdot, sk, z, z^*) \in \Lambda_{\mathcal{E}}$, we return $z$ otherwise
  - if $(sid_i^j, i, sk) \in \Lambda_{2\mathrm{P}}$, we choose $z \xleftarrow{\$} G^*$, if $(\cdot, \cdot, \cdot, \cdot, sk, z, \cdot) \in \Lambda_{\mathcal{E}}$, we stop the simulation and the adversary wins. Otherwise we return $z$ and add record $(dec, sid_i^j, i, \perp, sk, z, z^*)$ to $\Lambda_{\mathcal{E}}$.
  - if $(sid_i^j, i, sk) \notin \Lambda_{2\mathrm{P}}$, we choose $z \xleftarrow{\$} G^*$, if $(\cdot, \cdot, \cdot, \cdot, sk, z, \cdot) \in \Lambda_{\mathcal{E}}$, we stop the simulation and the adversary wins. Otherwise we return $z$ and add record $(dec, \perp, \perp, \perp, sk, z, z^*)$ to $\Lambda_{\mathcal{E}}$

This game is the same as the previous unless:

- Collision occurs in the simulation of encryption/decryption. This event happens with probability $\approx \frac{(q_{\mathcal{E}}+q_{\mathcal{D}})^2}{2|G|}$, where $q_{\mathcal{E}}$ is the number of encryptions and $q_{\mathcal{D}}$ is the number of decryptions.
- Value $sk$ had been first used by the decryption oracle $\mathcal{D}$ and then returned as a result of the 2P protocol in the first step of the protocol MLHL. This event occurs with probability $\frac{p(k)\cdot n\cdot q_{\mathcal{D}}}{2^{l_k}}$, where $p(\cdot)$ is a polynomial and $q_{\mathcal{D}}$ denotes number of decryptions ($p(k) \cdot n$ is number 2P's executions).

Hence,

$$|\Pr[Succ_0'] - \Pr[Succ_0]| \leq \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}}.$$

Next, in the games $G_1^i$ we simulate gradual replacement of values $sk_i$ by random keys. We alter the simulation of Execute and $\mathrm{Send}_2^1$ queries as follows: session key $sk_i$ established during 2P protocol between participant $P_i$ and server $S$ is replaced by a random string $sk_i'$, while we keep these randomly chosen values in the list $\Lambda_{2\mathrm{P}}$ in the format $(sid_i^j, i, sk_i')$. The randomly chosen values $sk_i'$ should not repeat for any participant and any session, if some $sk_i'$ is repeated, we stop the simulation and we let the adversary win (this happens with probability $\frac{p(k)^2}{2^{l_k+1}}$, where $p(k)$ specifies number of simulations of the MLHL protocol.
**Game** $G_1^1$:
In this game the session key established during 2P protocol between participant

$P_1$ and server $S$ is replaced by a random string $sk'_1$. In the list $\Lambda_{2\mathrm{P}}$ we store values $(sid_1^j, 1, sk'_1)$. We show that

$$|\Pr[Succ_1^1] - \Pr[Succ_0']| \leq p(k)\mathbf{Adv}^{\mathrm{AKE}}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}(t, q_{execute}, q_{send}, q_{reveal}) + \frac{p(k)^2}{2^{l_k+1}},$$

where $q_{send}, q_{execute}, q_{reveal}$ is number of Send, Execute, Reveal queries of 2P on his oracles and $p(\cdot)$ is polynomial.
To show this inequality we use a hybrid argument: we assume that there is a polynomial time distinguisher $\mathcal{D}$ that can distinguish games $G'_0$ and $G_1^1$ with probability $\varepsilon$:

$$|\Pr[\mathcal{D}^{G'_0} \to 1] - \Pr[\mathcal{D}^{G_1^1} \to 1]| = \varepsilon.$$

We show that if $\varepsilon$ is not negligible, we can construct an adversary $\mathcal{A}_{2\mathrm{P}}$ against AKE security of the 2P protocol, which probability of success is not negligible. We define sequence distributions $H_1^i$, $i = 0 \dots p(k)$, in distribution $H_1^i$ the first $i$ session keys established during 2P protocol between participant $P_1$ and server $S$ are replaced by a random string $sk'_1$. Clearly distribution $H_1^0$ is equal to game $G'_0$ and distribution $H_1^{p(k)}$ is equal to game $G_1^1$.
**Adversary** $\mathcal{A}_{2\mathrm{P}}$

1. $\mathcal{A}_{2\mathrm{P}}$ selects an index $j$ at random from $\{1 \dots p(k) - 1\}$ and a bit $b \xleftarrow{\$} \{0, 1\}$. $\mathcal{A}_{2\mathrm{P}}$ runs distinguisher $\mathcal{D}$ and responds to his oracle queries (described later). We assume that $\mathcal{A}_{2\mathrm{P}}$ is able to identify, which queries asked by $\mathcal{D}$ belong to the 2P protocol $(\mathrm{Send}_1^1, \dots, \mathrm{Send}_1^l)$ and which belong to the rest of the protocol MLHL $(\mathrm{Send}_2^1, \dots, \mathrm{Send}_4)$. $\mathcal{A}_{2\mathrm{P}}$ will simulate oracle queries of $\mathcal{D}$ as follows:
   - $\mathrm{Send}(\Pi_1^l, M)$ in 2P, $l < j$: $\mathcal{A}_{2\mathrm{P}}$ replies with the response of his $\mathrm{Send}(\Pi_1^l, M)$ oracle. If this query leads to establishment of a session key in 2P, then $\mathcal{A}_{2\mathrm{P}}$ selects a random key $sk'_1$ and uses it as a session key $sk_1$ between $P_1$ and $S$ in the session with $sid_1^l$.
   - $\mathrm{Send}(\Pi_1^j, M)$ in 2P: $\mathcal{A}_{2\mathrm{P}}$ replies with the response of his $\mathrm{Send}(\Pi_1^j, M)$ oracle. If this query leads to establishment of a session key in 2P, then $\mathcal{A}_{2\mathrm{P}}$ asks $\mathrm{Test}(\Pi_1^j)$ query and the result is used as a session key $sk_1$ between $P_1$ and $S$ in the MLHL protocol with session identifier $sid_1^j$.
   - $\mathrm{Send}(\Pi_i^l, M)$ in 2P, $i \neq 1 \wedge l \in \{1, \cdots p(k)\}$ or $i = 1 \wedge l > j$: $\mathcal{A}_{2\mathrm{P}}$ replies with the response of his $\mathrm{Send}(\Pi_i^j, M)$ oracle. If this query leads to establishment of a session key in 2P, then $\mathcal{A}_{2\mathrm{P}}$ asks $\mathrm{Reveal}(\Pi_i^l)$ query and the returned result is used as a session key $sk_i$ between $P_i$ and $S$ in the session with $sid_i^j$.
   - $\mathrm{Send}(\Psi^s, M)$ in 2P: similar as $\mathrm{Send}(\Pi_i^j, M)$
   - $\mathrm{Send}(\Pi_i^j, M)$ query outside 2P: $\mathcal{A}_{2\mathrm{P}}$ answers with the result of simulation of sending the message $M$ in MLHL, while he follows rules and steps of MLHL as in the previous game. During the simulation he uses keys $sk_i, i \in \{1, 2, ..., n\}$ (which were obtained as a response of his Reveal or Test oracle or by a random choice).

- Send($\Psi^s, M$) query outside 2P: similar to Send($\Pi_i^j, M$) outside 2P.
- Execute($P_1, P_2, \ldots, P_n, S$): If instance of $P_1$ has form $\Pi_1^l$, where $l = j$ then response to this Execute query is computed as follows: $\mathcal{A}_{2\mathrm{P}}$ starts to simulate 2P between participants $P_1, \ldots, P_n$ and $S$ with use of his own Execute oracle. Then he asks Test query on the instance of participant $P_1$ and Reveal queries on instances of participants $P_2, \ldots, P_n$ in that session. Next he continues with simulation of the rest of MLHL. Encryption and decryption of messages is done using the obtained keys $sk_i$.
  Other Execute queries, where $l \neq j$ are simulated in a different way.
  - If instance of $P_1$ has form $\Pi_1^l$, where $l < j$ then $\mathcal{A}_{2\mathrm{P}}$ starts to simulate 2P between participants $P_1, \ldots, P_n$ and $S$ with the use of his own Execute oracle. After simulation of 2P protocol he chooses key $sk_1'$ randomly and asks Reveal queries on all participants $P_i$, $i \geq 2$. Next he continues with the simulation of the rest of the MLHL protocol with obtained keys as in previous game.
  - If instance of $P_1$ has form $\Pi_1^l$, where $l > j$ then $\mathcal{A}_{2\mathrm{P}}$ starts to simulate 2P between participants $P_1, \ldots, P_n$ and $S$ with the use of his own Execute oracle, then he asks Reveal queries on all participants $P_i$, $i \geq 1$. Next he continues with the simulation of the rest of MLHL protocol with obtained keys as in previous game.
- Reveal($\Pi_i^j$): $\mathcal{A}_{2\mathrm{P}}$ answers under the rules of Reveal query in the security model (he returns the real session key $sk$, if $\Pi_i^j$ has the key established during the simulation)
- Test($\Pi_i^j$): if the randomly chosen bit $b = 0$, $\mathcal{A}_{2\mathrm{P}}$ returns the real session key $sk$ (computed during the simulation of Send or Execute queries), otherwise he returns a randomly chosen key $sk'$.

2. $\mathcal{A}_{2\mathrm{P}}$ returns $b \leftarrow \mathcal{D}$

Now we analyze the behaviour of $\mathcal{A}_{2\mathrm{P}}$. Fix polynomial $p(\cdot)$ and $\mathcal{A}_{2\mathrm{P}}$ chooses $j = J$, where $J$ is random value uniformly chosen from $\{1, \ldots, p(k)\}$. If $\mathcal{A}_{2\mathrm{P}}$ during $\mathrm{GameAKE}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}$ gets real session key established during protocol 2P between participants $P_1$ and $S$, then the view of distinguisher $\mathcal{D}$ is as in distribution $H_1^{J-1}$. That is,

$$\Pr_{sk_1 \leftarrow 2\mathrm{P}(P_1,S)}[\mathcal{A}_{2\mathrm{P}}(sk_1) = 1 | j = J] = \Pr_{view \leftarrow H_1^{J-1}}[\mathcal{D}(view) = 1].$$

Since the value of $j$ is chosen uniformly at random, we have

$$\begin{aligned}\Pr_{sk_1 \leftarrow 2\mathrm{P}(P_1,S)}[\mathcal{A}_{2\mathrm{P}}(sk_1) = 1] &= \frac{1}{p(k)} \sum_{J=1}^{p(k)} \Pr_{sk_1 \leftarrow 2\mathrm{P}(P_1,S)}[\mathcal{A}_{2\mathrm{P}}(sk_1) = 1 | j = J] \\ &= \frac{1}{p(k)} \sum_{J=1}^{p(k)} \Pr_{view \leftarrow H_1^{J-1}}[\mathcal{D}(view) = 1].\end{aligned}$$

If $\mathcal{A}_{2\mathrm{P}}$ chooses $j = J$ and during $\mathrm{GameAKE}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}$ it receives a randomly chosen value instead of the session key as a response of its Test oracle, then the view of distinguisher $\mathcal{D}$ is as in distribution $H_1^J$. That is,

$$\Pr_{sk_1' \leftarrow \{0,1\}^{l_k}}[\mathcal{A}_{2\mathrm{P}}(sk_1') = 1 | j = J] = \Pr_{view \leftarrow H_1^J}[\mathcal{D}(view) = 1].$$

Then, we have

$$\begin{aligned}\Pr_{sk_1' \leftarrow \{0,1\}^{l_k}}[\mathcal{A}_{2\mathrm{P}}(sk_1') = 1] &= \frac{1}{p(k)} \sum_{J=1}^{p(k)} \Pr_{sk_1' \leftarrow \{0,1\}^{l_k}}[\mathcal{A}_{2\mathrm{P}}(sk_1') = 1 | j = J] \\ &= \frac{1}{p(k)} \sum_{J=1}^{p(k)} \Pr_{view \leftarrow H_1^J}[\mathcal{D}(view) = 1].\end{aligned}$$

In the end we have

$$\begin{aligned}&\Big| \Pr_{sk_1' \leftarrow \{0,1\}^{l_k}}[\mathcal{A}_{2\mathrm{P}}(sk_1') = 1] - \Pr_{sk_1 \leftarrow 2\mathrm{P}(P_1,S)}[\mathcal{A}_{2\mathrm{P}}(sk_1) = 1] \Big| \\ &= \frac{1}{p(k)} \Big| \sum_{J=1}^{p(k)} \Pr_{view \leftarrow H_1^J}[\mathcal{D}(view) = 1] - \sum_{J=0}^{p(k)-1} \Pr_{view \leftarrow H_1^J}[\mathcal{D}(view) = 1] \Big| \\ &= \frac{1}{p(k)} \Big| \Pr_{view \leftarrow H_1^{p(k)}}[\mathcal{D}(view) = 1] - \Pr_{view \leftarrow H_1^0}[\mathcal{D}(view) = 1] \Big| = \frac{\varepsilon}{p(k)}.\end{aligned}$$

Since 2P is AKE secure protocol and $\mathcal{A}_{2\mathrm{P}}$ runs in polynomial time and $p(\cdot)$ is polynomial, the value $\varepsilon$ must be negligible.

$$\begin{aligned}|\Pr[Succ_1^1] - \Pr[Succ_0']| &= | \Pr_{view \leftarrow H_1^{p(k)}}[\mathcal{D}(view) = 1] - \Pr_{view \leftarrow H_1^0}[\mathcal{D}(view) = 1]| \\ &\leq p(k)\mathbf{Adv}^{\mathrm{AKE}}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}(t, q_{execute}, q_{send}, q_{reveal}) + \frac{p(k)^2}{2^{l_k+1}}.\end{aligned}$$

**Game** $G_1^2$:
In this game we change the previous simulation so that the established session key between the participant $P_2$ and the server $S$ is replaced by a random string $sk_2'$. Thus session keys between the participants $P_1, P_2$ and the server $S$ are randomly generated. The similar reasoning of existence of a distinguisher between games $G_1^1$ and $G_1^2$ works. Therefore we have

$$|\Pr[Succ_1^2] - \Pr[Succ_1^1]| \leq p(k)\mathbf{Adv}^{\mathrm{AKE}}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}(t, q_{execute}, q_{send}, q_{reveal}) + \frac{p(k)^2}{2^{l_k+1}}.$$

The games $G_1^3, \ldots, G_1^n$ are defined similarly. When we sum all inequalities on the left side and on the right side,

$$|\Pr[Succ_1^n] - \Pr[Succ_0']| \leq n \cdot p(k)\mathbf{Adv}^{\mathrm{AKE}}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}(t, q_{execute}, q_{send}, q_{reveal}) + \frac{np(k)^2}{2^{l_k+1}}.$$

In this part we simulate gradual replacement of values $K_i$ by random values in the games $G_2^i, i = 1 \dots n$. We alter the simulation of Execute queries as follows: a Diffie-Hellman value $K_i$ established during the MLHL protocol between participants $P_i$ and $P_{i+1}$ is replaced by a random value $K_i'$ from $G$.

**Game** $G_2^1$: In this game we simulate everything like in the previous game, however the value $K_1$ is replaced by a random value during Execute queries. We show that

$$|\Pr[Succ_2^1] - \Pr[Succ_1^n]| \leq p(k)\mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'').$$

To prove this inequality, suppose that there exist a distinguisher $\mathcal{D}$ which can distinguish these two games. We can use this distinguisher to construct an adversary $\mathcal{A}_{\mathrm{DDH}}$, which can solve DDH problem, with use of similar hybrid argument as in previous games: we define distribution $H_2^i$, $i \in \{0, 1, \dots p(k)\}$. In the distribution $H_2^i$ the values $K_1$ for instances $\Pi_1^j$, $j \leq i$ are chosen randomly and the values $K_1$ for instances $\Pi_1^j$, $j > i$ are computed as in the previous game. We assume that distinguisher $\mathcal{D}$ during simulation constructs $p(k)$ sessions for some polynomial $p(\cdot)$.

**Adversary** $\mathcal{A}_{\mathrm{DDH}}(u, v, w)$

1. $\mathcal{A}_{\mathrm{DDH}}$ chooses random bit $b$ and index $j$.
2. $\mathcal{A}_{\mathrm{DDH}}$ answers oracle queries of the distinguisher $\mathcal{D}$ as follows:
   - Send, Reveal and Test queries are answered as in previous game, Test queries are answered with the use of bit $b$ (note, that the adversary knows the established session keys, because he simulated the execution).
   - Execute$(P_1, \dots, P_n)$ queries are simulated in the following way:
     If instance of $P_1$ has form $\Pi_1^l$, where $l = j$ then simulation of Execute query for instances of participants $P_3, \dots, P_n$ in the same session does not change. Protocol 2P between $P_1$, $P_2$ and $S$ is simulated as in the previous game, after this simulation $\mathcal{A}_{\mathrm{DDH}}$ knows the values $sk_1, sk_2$ – he has chosen them randomly. Then he simulates that $P_1$ sends the message $(ID(P_1)||\mathcal{E}_{sk_1}(u))$ and $P_2$ sends the message $(ID(P_2)||\mathcal{E}_{sk_2}(v))$ then $K_1$ is set to $w$, $K_2 = v^{x_3}$, $K_n = u^{x_n}$. Next he continues with the simulation of the rest of MLHL. Other Execute queries, where $l \neq j$ are simulated as follows:
     - If instance of $P_1$ has form $\Pi_1^l$, where $l < j$ then $\mathcal{A}_{\mathrm{DDH}}$ starts to simulate 2P between participants $P_1, \dots, P_n$ and $S$ as in previous game. After simulation of 2P protocol he chooses randomly keys $sk_i'$, $i = 1 \dots n$ and simulates the rest of the MLHL as in the previous game however, the computed value $K_1$ in each session is replaced by random value.
     - If instance of $P_1$ has form $\Pi_1^l$, where $l > j$ then $\mathcal{A}_{\mathrm{DDH}}$ starts to simulate 2P between participants $P_1, \dots, P_n$ and $S$ as in previous game. After simulation of 2P protocol he continues with simulation of the rest of the MLHL as in the previous game.
3. $\mathcal{A}_{DDH}$ returns $b \leftarrow \mathcal{D}$

If $(u, v, w)$ from adversary's input is a DDH triple and index $j = 0$, the view of distinguisher $\mathcal{D}$ is the same as in the game $G_1^n$ ($H_2^0$). If $(u, v, w)$ is not a DDH

triple and index $j = p(k)$, the view of $\mathcal{D}$ is the same as in the game $G_2^1$ ($H_2^{p(k)}$). Thus the advantage of $\mathcal{A}_{\mathrm{DDH}}$ is at least as great as $\frac{1}{p(k)}$ of the advantage of $\mathcal{D}$ (we skip the detailed reasoning).

$$|\Pr[Succ_2^1] - \Pr[Succ_1^n]| = p(k)\mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'')$$

**Game** $G_2^2$:
In this game we change previous simulation that the computed value $K_2$ is replaced by a random value. Therefore values $K_1, K_2$ are randomly generated. The similar reasoning about existence of a distinguisher between games $G_2^1$ and $G_2^2$ works. Therefore we have

$$|\Pr[Succ_2^2] - \Pr[Succ_2^1]| = p(k)\mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'').$$

The games $G_2^3, \ldots, G_2^n$ are defined similarly. When we sum inequalities, we have

$$|\Pr[Succ_2^n] - \Pr[Succ_1^n]| = n \cdot p(k) \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'').$$

**Game** $G_3$:
In this game the session key of MLHL is replaced by a random value during Execute queries. We have

$$\Pr[Succ_3] = \Pr[Succ_2^n].$$

This claim follows from the view of an adversary in this two games. In the game $G_2^n$ the values $K_i$ are chosen at random, therefore they are independent from previously sent messages (They are not sent directly, but as xor-ed values $w_i$, which can originate from combination of $2^{|w_i|}$ different pairs of values). This implies that the computed $w_i$ (which adversary can see) are independent from previously sent messages. From all of this facts follows, that the computed session key is independent from all sent values and therefore there is no difference between these games.
**Game** $G_4$:
In this game we change simulation of the first subcase of decryption oracle (defined in game $G_0'$) in Send queries: we build an instance of PDDH problem in simulation of the protocol. We set $\beta = 0$, thus the challenger $Chall^{\beta}(\cdot)$ returns vectors $(\zeta_1, ..., \zeta_n, \gamma_1, ..., \gamma_n)$ from the distribution $PDH_n^*$. New vectors are returned in every session, however the same vectors are returned in queries on the same session. For randomly chosen $(\alpha_1, ..., \alpha_n)$, $\alpha_i \xleftarrow{\$} Z_q^*$ have vectors $(\zeta_1^{\alpha_1}, ..., \zeta_n^{\alpha_n}, \gamma_1^{\alpha_1\alpha_2}, ..., \gamma_n^{\alpha_n\alpha_1})$ equal distribution to the original $(\zeta_1, ..., \zeta_n, \gamma_1, ..., \gamma_n)$. We use this property for application of random self-reducibility of the PDDH problem. The decryption is changed as follows:

- $\mathcal{D}_{sk_i}(z^*)$ – if $(sid_i^j, i, sk_i) \in \Lambda_{2\mathrm{P}}$, $(\zeta_1, ..., \zeta_n, \gamma_1, ..., \gamma_n) \leftarrow Chall^{\beta}(sk_1, ..., sk_n)$ (the arguments of $Chall^{\beta}$ can be found in the $\Lambda_{2\mathrm{P}}$ list sharing the same value of session ID), we choose random $\alpha_i \xleftarrow{\$} Z_q^*$ and compute $z_i = \zeta_i^{\alpha_i}$. If $(\cdot, \cdot, \cdot, \cdot, sk_i, z_i, \cdot) \in \Lambda_{\mathcal{E}}$, then we stop the simulation, adversary wins. Otherwise we add record $(dec, sid_i^j, i, \alpha_i, sk_i, z_i, z^*)$ to $\Lambda_{\mathcal{E}}$ and return $z_i$.

Exponent $\alpha_i$ specifies, how we applied random self-reducibility of PDDH problem on instance generated by the challenger. Exponent $\alpha_i$ can be defined in the list $\Lambda_{\mathcal{E}}$ only if values $sid_i^j$ and $i$ are known. The view of the adversary does not change and therefore we have

$$\Pr[Succ_4] = \Pr[Succ_3].$$

**Game** $G_5$:
We change the simulation of $\text{Send}_2^1$, $\text{Send}_2^2$ and $\text{Send}_3$ queries. First, encryption of messages in the second step of the protocol is changed during simulation of $\text{Send}_2^1$. Instance $\Pi_i^j$ chooses $z_i^* \xleftarrow{\$} G$ randomly and computes $z_i = \mathcal{D}_{sk_i}(z_i^*)$ as in the previous game. Then $\Pi_i^j$ sends the message $(ID(P_i)||z_i^*)$. Therefore $\text{Send}_2^1$ queries in the second step of the MLHL lead to adding of $\alpha_i$ to the list $\Lambda_{\mathcal{E}}$. Simulation ends if

- $(enc, \perp, \perp, \perp, sk_i, \cdot, z_i^*) \in \Lambda_{\mathcal{E}}$, because we do not know the value of $\alpha_i$. This possibility occurs if the adversary asks for encryption of some value with the key $sk_i$ and the result of encryption was $z_i^*$ (it means that $(enc, \perp, \perp, \perp,$ $sk_i, \cdot, z_i^*) \in \Lambda_{\mathcal{E}}$). The probability of this event is $q_{\mathcal{E}}/2^{l_k}$. In this case we stop the simulation, the adversary wins.

- $(dec, \perp, \perp, \perp, sk_i, z_i, z_i^*) \in \Lambda_{\mathcal{E}}$. This possibility occurs if we decrypt the value $z_i^*$, while the values $i, sid_i^j$ belonging to $sk_i$ were not known. However this situation can not occur (see Game $G_0'$, point 3).

When the server accepts the message $(ID(P_i)||z_i^*)$ during simulation of $\text{Send}_2^2$, he should resend it to participants $P_{i-1}$ and $P_{i+1}$, thus he must decrypt $z_i^*$. The following cases can occur:

- $z_i^*$ was encrypted in the aforementioned manner, thus we know the value $\alpha_i$. We can continue with the simulation of encryption described bellow.
- $z_i^*$ is response of encryption oracle $\mathcal{E}_{sk_i}$, while $sk_i$ is correct key of $\Pi_i^j$ in the corresponding session (thus adversary guessed the $sk_i$ and used it for encryption of data for server). In this case we stop the simulation, adversary wins. This event occurs with probability $q_{\mathcal{E}}/2^{l_k}$.
- $z_i^*$ was chosen by adversary without asking encryption oracle. In this situation adversary does not know the password and therefore he could not compute messages in the way, that they go through the control step. Simulation continues as follows: we compute $z_i' = \mathcal{D}_{sk_i}(z_i^*)$, then we compute $z_i^{**} = \mathcal{E}_{sk_{i+1}}(z_i')$ and send to the user $P_{i+1}$ message $(ID(S)||ID(P_i)||z_i^{**})$. Similar for $P_{i-1}$. Next we simulate as in the previous games, however, adversary does not know any of values $K_i$, therefore he could not manipulate other messages in the way, that they go through the verification step of MAC scheme, unless he breaks it with probability $\mathbf{Adv}_{\text{M},\mathcal{A}_{\text{M}}}^{\text{MAC}-\text{forge}}(t')$, which is negligible.

Encryption of $z_i = \mathcal{D}_{sk_i}$ (in the first case) with another passwords ($sk_{i-1}$ and $sk_{i+1}$) (in second step) works as follows:

- $\mathcal{E}_{sk_{i+1}}(z_i)$
  - if $(\cdot,\cdot,\cdot,\cdot,sk_{i+1},z_i,\cdot) \notin \Lambda_{\mathcal{E}}$ and $(dec, sid_i^j, i, \alpha_i, sk_i, z_i, \cdot) \in \Lambda_{\mathcal{E}}$ (this record was added in simulation described above by the instance $\Pi_i^j$), then we choose $z^{**} \xleftarrow{\$} G$ if $(\cdot,\cdot,\cdot,\cdot,sk_{i+1},\cdot,z^{**}) \notin \Lambda_{\mathcal{E}}$, we return $z^{**}$ and add record $(enc, sid_i^j, i, \perp, sk_{i+1}, z, z^{**})$ into $\Lambda_{\mathcal{E}}$, else we stop the simulation and adversary wins.
  - if $(enc, \perp, \perp, \perp, sk_{i+1}, z_i, \cdot) \in \Lambda_{\mathcal{E}}$, we stop the simulation and the adversary wins. This case occurs if adversary asked for encryption of value $z_i$ with key $sk_{i+1}$. The probability of this event is $q_{\mathcal{E}}/2^{l_k}$.
  - if $(enc, sid_i^j, i+2, \perp, sk_{i+1}, z_i, z^*) \in \Lambda_{\mathcal{E}}$, we return $z^*$. This case occurs if during the simulation of execution of the protocol happens request for resending the value $z_i$ to instance $\Pi_{i+1}^j$ (sent with instance $\Pi_{i+2}^j$), while the value was encrypted with the key $sk_{i+1}$.
  - if $(dec, sid_i^j, i+1, \alpha_{i+1}, sk_{i+1}, z_i, z^*) \in \Lambda_{\mathcal{E}}$, we return $z^*$. This case can occur by simulation of Send queries of instance $\Pi_{i+1}^j$ in second round.
- $\mathcal{E}_{sk_{i-1}}(z_i)$ – similar to the previous case.

Simulation of $\text{Send}_3$, when $\Pi_i^j$ receives messages $(ID(S)||ID(P_{i-1})||z_{i-1}^*)$ and $(ID(S)||ID(P_{i+1})||z_{i+1}^*)$ works as follows: compute $z_{i-1} = \mathcal{D}_{sk_i}(z_{i-1}^*)$ and $z_{i+1} = \mathcal{D}_{sk_i}(z_{i+1}^*)$. This three cases can occur:

- $z_{i-1}^*$ and $z_{i+1}^*$ were encrypted in the previous manner. We can continue with simulation as described bellow.
- one or both $z_{i-1}^*$ and $z_{i+1}^*$ is/are the answer from query on encryption oracle $\mathcal{E}_{sk_i}$, while $sk_i$ is correct key of instance $\Pi_i^j$ in session $j$ (thus adversary guessed password and used him for encryption of data from server). This event occurs with probability $q_{\mathcal{E}}/2^{l_k}$. In this case we stop the simulation, adversary wins.
- one or both $z_{i-1}^*$ and $z_{i+1}^*$ was/were chosen by adversary without asking of Encryption oracle, in this situation adversary do not know the password and therefore he should not compute messages in the way, that they go through the verification step of MAC scheme, unless he breaks it with probability $\mathbf{Adv}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}^{\mathrm{MAC-forge}}(t')$, which is negligible.

If the messages were sent as we simulate them, we have:

$$z_i = \zeta_i^{\alpha_i}, z_{i-1} = \zeta_{i-1}^{\alpha_{i-1}}, z_{i+1} = \zeta_{i+1}^{\alpha_{i+1}}$$

and we can compute

$$K_{i-1} = \mathcal{H}(CDH(z_{i-1}, z_i)), K_i = \mathcal{H}(CDH(z_i, z_{i+1}))$$

$$w_i = K_{i-1} \oplus K_i, \tau_i = \mathrm{Mac}_{K_i}(w_i)$$

and resend the message $(ID(P_i), w_i, \tau_i)$. When every participant broadcasts such message, the session key can be computed.
This game is the same as the previous unless mentioned "bad" events happen.

$$|\Pr[Succ_5] - \Pr[Succ_4]| \leq 4q_{\mathcal{E}}/2^{l_k} + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}(t')$$

**Game** $G_6$:
In this game we change the bit $\beta$ to 1, thus the values $(\zeta_1, ..., \zeta_n, \gamma_1, ..., \gamma_n)$ are from distribution $PDH^{\$}_n$. Clearly holds that

$$|\Pr[Succ_6] - \Pr[Succ_5]| \leq p(k) \cdot \mathbf{Adv}^{\mathrm{PDDH}_n}_{G,\mathcal{A}_{\mathrm{PDDH}}}(t''),$$

where $p(k)$ is the number of sessions, $p$ is a polynomial.
**Game** $G_7$:
The session key of MLHL is replaced by a random value during Send queries in this game. We have

$$\Pr[Succ_7] = \Pr[Succ_6].$$

This claim follows from the view of adversary in this two games. In game $G_6$ are values $K_i$ chosen at random, therefore they are independent from previous sent messages (They are not sent directly, but as xor-ed values $w_i$, which can originate from combination of $2^{|w_i|}$ different pairs of values). This implies that the computed $w_i$ (which adversary sees) are independent from previous sent messages. From all of this facts follows, that the computed session key is independent from all sent values and therefore there is no difference between these games.
The probability of adversary's success in this game is $\frac{1}{2}$, because the session key is randomly chosen and independent from the previous messages. Therefore

$$\Pr[Succ_7] = \frac{1}{2}.$$

When we sum all (in)equalities of games, we have:

$$\begin{aligned}|\Pr[Succ_7] - \Pr[Succ_0]| \leq{}& \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}(t') \\ &+ p(k) \cdot n\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{2P},\mathcal{A}_{\mathrm{2P}}}(t, q_{execute}, q_{send}, q_{reveal}) \\ &+ \frac{np(k)^2}{2^{l_k+1}} + p(k) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'') + 4q_{\mathcal{E}}/2^{l-k} \\ &+ p(k) \cdot \mathbf{Adv}^{\mathrm{PDDH}_n}_{G,\mathcal{A}_{\mathrm{PDDH}}}(t'')\end{aligned}$$

$$\begin{aligned}\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}(\mathcal{A}) \leq 2\Bigg(& \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}(t') \\ &+ p(k) \cdot n\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{2P},\mathcal{A}_{\mathrm{2P}}}(t, q_{execute}, q_{send}, q_{reveal}) \\ &+ \frac{np(k)^2}{2^{l_k+1}} + 2p(k) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'') + 4q_{\mathcal{E}}/2^{l-k}\Bigg).\end{aligned}$$

### A.2 Advantage of adversary in $Case_2$:

In this section we prove the AKE-fs security of protocol in $Case_2$, where the adversary knows all LL-keys, if he asks Corrupt queries for all participants $P_i$, $i = \{1, \ldots, n\}$ after Test query (notice, that means, that instance on which adversary makes Test query has established session key before any Corrupt query was made). He can make Send, Execute, Reveal and Test queries. Also he can check, if the session keys of 2P protocol were changed: he can compare, if encrypted part of sent message in second step after decryption, which was sent by $P_i$ is the same like encrypted part of sent message in second step $\mathcal{D}_{sk_i}(\mathcal{E}_{sk'_i}(g^{x_i})) = \mathcal{D}_{sk_{i+1}}(\mathcal{E}_{sk_{i+1}}(g^{x_i}))$

**Lemma 2.** *The advantage of the adversary from $Case_2$ is:*

$$\boldsymbol{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}},Case_2}(T) \leq 2\Bigg(\frac{(q_{\mathcal{E}}+q_{\mathcal{D}})^2}{2|G|} + 4q_{\mathcal{E}}/2^{l-k}\frac{p(k)\cdot n\cdot q_{\mathcal{D}}}{2^{l_k}} + \varepsilon$$
$$+ 2p(k)\cdot n\cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'') + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}(t')\Bigg).$$

*Proof.* Similar like in the previous case, we start with the simulation of the real protocol.
**Game** $G_0$:
This is a game with the real protocol and from definition we have:

$$\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}(T) = 2\Pr[Succ_0] - 1.$$

**Game** $G'_0$:
Is same as in the $Case_1$. Therefore

$$|\Pr[Succ'_0] - \Pr[Succ_0]| \leq \frac{(q_{\mathcal{E}}+q_{\mathcal{D}})^2}{2|G|} + \frac{p(k)\cdot n\cdot q_{\mathcal{D}}}{2^{l_k}}.$$

**Game** $G_1$:
Games $G^i_1$ from the $Case_1$ are not possible, because the adversary can check, if the session keys of 2P were changed. In this game we keep all session keys established in 2P in a list $\Lambda_{2\mathrm{P}}$ in format $(sid^j_i, i, sk_i)$, where $sid^j_i$ is a session ID, $i$ is index of $P_i$ and $sk_i$ is a session key established in the protocol 2P between $P_i$ and $S$. If during some simulation an arbitrary participant establishes with server session key, which is in $\Lambda_{2\mathrm{P}}$, we stop the simulation, adversary wins (session keys can not repeat). This event happens with negligible probability $\varepsilon$. If $\varepsilon$ is not negligible, we have conflict with assumption that 2P is AKE secure 2PAKE protocol, because the distribution of session key is not random.

$$|\Pr[Succ_1] - \Pr[Succ_0]| \leq \varepsilon$$

The games $G_2^1, \ldots, G_2^n$ are defined as in the previous case. Therefore we have

$$|\Pr[Succ_2^n] - \Pr[Succ_1]| \leq n \cdot p(k) \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'').$$

The games $G_3$, $G_4$, $G_5$, $G_6$ and $G_7$ are simulated as in the previous case. When we sum all (in)equalities of games, we have:

$$\begin{aligned}|\Pr[Succ_7] - \Pr[Succ_0]| \leq \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + \varepsilon + 4q_{\mathcal{E}}/2^{l-k} \\ + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}} + p(k) \cdot \mathbf{Adv}^{\mathrm{PDDH}_n}_{G,\mathcal{A}_{\mathrm{PDDH}}}(t'') \\ + p(k) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'')\end{aligned}$$

$$\begin{aligned}\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}},Case_2}(T) \leq 2\Bigg(\frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + 4q_{\mathcal{E}}/2^{l-k}\frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + \varepsilon \\ + 2p(k) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'') + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}(t')\Bigg).\end{aligned}$$

### A.3 Advantage of adversary in $Case_3$:

In this section we prove the AKE-fs security of protocol in $Case_3$, the adversary can make only Reveal, Execute queries on instances on which he wants to ask Test query.

**Lemma 3.** *The advantage of the adversary from $Case_3$ is:*

$$\begin{aligned}\boldsymbol{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}},Case_3}(T) \leq 2\Bigg(\frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + \varepsilon \\ + p(k) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'')\Bigg).\end{aligned}$$

The proof of this lemma is very similar as in $Case_2$.

*Proof.* Similar like in the previous case, we start with the simulation of the real protocol.
**Game** $G_0$:
This is a game with the real protocol and from definition we have:

$$\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}},Case_3}(T) = 2\Pr[Succ_0] - 1.$$

**Game** $G_0'$:
Is same as in the $Case_2$. Therefore

$$|\Pr[Succ_0'] - \Pr[Succ_0]| \leq \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}}.$$

**Game** $G_1$:
Is same as in the $Case_2$. We have

$$|\Pr[Succ_1] - \Pr[Succ_0]| \leq \varepsilon$$

The games $G_2^1, \ldots, G_2^n$ are defined as in the previous case. Therefore we have

$$|\Pr[Succ_2^n] - \Pr[Succ_1]| \leq n \cdot p(k) \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'').$$

**Game** $G_3$:
The session key of MLHL is replaced by a random value during Execute queries in this game. We have

$$\Pr[Succ_3] = \Pr[Succ_2^n].$$

This claim follows from the view of adversary in this two games. In the game $G_2^n$ are values $K_i$ chosen at random, therefore they are independent from previous sent messages (They are not sent directly, but as xor-ed values $w_i$, which can originate from combination of $2^{|w_i|}$ different pairs of values). This implies that the computed $w_i$ (which adversary sees) are independent from previously sent messages. From all of this facts follows, that the computed session key is independent from all sent values and therefore there is not any difference between these games.
The probability of adversary's success in this game is $\frac{1}{2}$, because the session key is randomly chosen and independent from the previous messages. Therefore

$$\Pr[Succ_3] = \frac{1}{2}.$$

When we sum all (in)equalities of games, we have:

$$\begin{aligned}|\Pr[Succ_3] - \Pr[Succ_0]| \leq \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + \varepsilon \\ + p(k) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'')\end{aligned}$$

$$\begin{aligned}\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}},Case_3}(T) \leq 2\Bigg(\frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(k) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + \varepsilon \\ + p(k) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t'')\Bigg).\end{aligned}$$