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 the MLHL (Multi-LHL) protocol, which is based on the LHL protocol proposed by Lee, Hwang and Lee. However, the 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 the LHL protocol.

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 the 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 the 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 [BMP00, Ja96, GL01, KOY01].

Although the original idea of PAKE protocol EKE [BM92] 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 the case when one of the participants is compromised, the whole group needs to choose a new password. Instead, the schemes with single password per user offer much better user experience. However, this 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 against 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 the EKE (Encrypted key exchange) protocol based on Diffie-Hellman key exchange. EKE was proposed by Bellowin and Merritt in [BM92]. 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. Therefore, they define a security model for PAKE protocols and prove the correctness of EKE. Boyko, MacKenzie and Patel [BMP00] 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.

Kwon, Jeong, Sakurai and Lee [Kw06] deal with a multi-party scenario with a trusted server where each participant owns a different password. The goal of their protocols PAMKE_1 and 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 [LHL04]. The LHL protocol is however not secure as showed by Abdalla, Bresson and Chevassut in [ABC06] where they propose a new protocol secure against this attack. Choo [Ch06] suggested another attack on the LHL protocol.

In [Ab11] suggested construction that is secure in a common reference string, therefore it does not rely on any idealized model. They prove the security of construction in the universally composable framework.

Hao and Ryan [HR11] suggested a protocol, where two participants send ephemeral public keys to each other. Then they encrypt the password by juggling the public keys in a verifiable way.

Our contribution. We were inspired by the LHL protocol [LHL04]. However in [ABC06, Ch06] 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 [BM92, AFP05, BPR00, Kw06]. Our construction is secure against the attacks from [ABC06, Ch06]. Secondly, every participant has his own secret password (compared to the protocol suggested in [ABC06]) and because of this, there are no problems with adding a new participant and with compromising some participant. On the other hand, this requires a help of a server, which knows the password of each participant. When the server knows the passwords, it could try to learn the session key (because it is curious). Therefore we want to have a protocol in which server could not learn established session key from knowledge of passwords and the communication it sees. Our main contribution is the proof of security (denoted as AKE-fs, see Definition 8) 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 R from a finite set T where the element R is chosen uniformly is denoted as $R \xleftarrow{\$} T$. By $M_1 \parallel M_2$ we denote *concatenation* of two strings M_1 and M_2. Random oracle is a function $f : Y_1 \to Y_2$ uniformly chosen from the set $Func(Y_1, Y_2)$ of all functions with domain Y_1 and range Y_2. 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 is denoted as $E = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ and message authentication code scheme (MAC) is denoted as $M = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrf})$. A tag τ is computed as $\tau = \mathrm{Mac}_k(Msg)$ for message Msg with use of key k.

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 $\mathcal{C} = \{P_1, P_2, \ldots P_n\}$ 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 Π_i^j and $ID(P_i)$ is a unique identifier of participant P_i (analogously j-th instance of server S is denoted as Ψ^j). A group of participants

$P_{i_1}, P_{i_2}, \ldots, P_{i_k}$ is denoted as $Grp_{i_1,i_2,\ldots,i_k}$.

Definition 1. [BR95] A protocol is a triple $\mathcal{P} = (\Pi, \Psi, LL)$, where Π specifies how each client behaves, Ψ specifies how server behaves and LL specifies the distribution of long-lived keys.

Definition 2. 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 CON be a cryptographic construction (algorithm), $\mathcal{A}$ be an adversary and xxx be any problem on CON (such as collision resistance of hash function, or discrete logarithm in a group G). $\mathbf{Adv}^{\text{xxx}}_{CON,\mathcal{A}}$ is a measure of adversary's advantage defined as a probability, that $\mathcal{A}$ succeeds to solve the problem xxx for CON. 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 $\kappa_1, \kappa_2, \ldots \kappa_n$ be parameters needed for the security definition, then the adversary's advantage is denoted as $\mathbf{Adv}^{\text{xxx}}_{CONN,\mathcal{A}}(\kappa_1, \kappa_2, \cdots \kappa_n)$.

In this paper we adopt a Dolev-Yao model of an adversary, where the adversary intercepts whole communication during the execution of a protocol. The adversary can delay, change or deliver messages out of order, start a new execution of a protocol, acquire a LL-key of some participants and acquire a given session key. All abilities of the adversary are modelled through oracles defined in Section 3.

We use the notion of Parallel Decisional Diffie-Hellman assumption and a challenger $Chall^{\beta}(\cdot)$ defined by Abdalla et al. [ABC06] in our security proofs.

Definition 3 (Parallel Decisional Diffie-Hellmann assumption – PDDH$_n$)**.** Let G be a cyclic group of order q with generator g and $\mathcal{A}_D$ be an adversary (distinguisher). Two distributions are defined:
$PDH^*_n = \{(g^{x_1}, g^{x_2}, \ldots, g^{x_n}, g^{x_1x_2}, g^{x_2x_3}, \ldots, g^{x_nx_1}) | x_1, x_2, \ldots, x_n \xleftarrow{\$} Z^*_q\}$ 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 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 PDDH$_n$ assumption holds in a cyclic group G if and only if the advantage of every $\mathcal{A}_D$ on PDDH$_n$ problem in time t_{DDH} is negligible. This advantage is denoted as $\mathbf{Adv}^{\text{PDDH}_n}_{G,\mathcal{A}_D}(t_{DDH})$ and computed as:

$$\mathbf{Adv}^{\text{PDDH}_n}_{G,\mathcal{A}_D}(t_{DDH}) = |\Pr[\mathcal{A}_D(PDH^*_n) \to 1] - \Pr[\mathcal{A}_D(PDH^{\$}_n) \to 1]|.$$

In [ABC06], it was proved that for a group G, time t_{DDH}, an integer $n > 2$ and adversary $\mathcal{A}_D$ the PDDH$_n$ and DDH problems are equivalent in G:

$$\mathbf{Adv}^{\text{DDH}}_{G,\mathcal{A}_D}(t_{DDH}) \leq \mathbf{Adv}^{\text{PDDH}_n}_{G,\mathcal{A}_D}(t_{DDH}) \leq n \cdot \mathbf{Adv}^{\text{DDH}}_{G,\mathcal{A}_D}(t_{DDH})$$

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

3 Security model

In this section we present a model based on [BM92], later extended in [BPR00] and adapted for group key exchange in [Kw06]. For identification of concrete session and instance of a partner in the session we defined notions *session identifier* and *partnering*.

Definition 4. A session identifier (sid) is a unique identifier of a session. It is the same for all participants in the session. The session identifier of the instance Π_i^j is denoted as sid_i^j. For the server instance Ψ^s is the session identifier denoted as sid^s.

If instances Π_i^j, Π_k^l and Ψ^s are in the same session, then $sid_i^j = sid_k^l = sid^s$.

Definition 5. A partner identifier pid_i^j for the instance Π_i^j is set of all identifiers of instances with whom Π_i^j wants to establish a session key. Instances Π_i^j and Π_k^l are partners, if

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

The adversary controls whole communication. He can stop sent message, send message Msg, deliver messages out of order and intercept communication. His abilities are modelled using the following oracles:

- Send(Π_i^j, Msg) – sends the message Msg to the instance Π_i^j in the session sid_i^j and returns a reply of Π_i^j (according to the execution of the protocol). This oracle query simulates an active attack of the adversary.
- Send(Ψ^s, Msg) – similarly to the $Send(\Pi_i^j, Msg)$. This oracle query sends the message Msg to the instance of the server Ψ^s in the session sid^s and returns a reply of Ψ^s.
- Execute$(Grp_{i_1,i_2,\ldots,i_k}, S)$ – this oracle starts execution of a protocol between participants $P_{i_1}, P_{i_2}, \ldots, P_{i_k}$ and the 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(Π_i^j) – if the instance Π_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 the 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 the participant P_i too.)
- Test(Π_i^j) – This query can be used only on a fresh/fs-fresh instance (see Def. 6). First a random bit $b \xleftarrow{\$} \{0,1\}$ is chosen. If instance Π_i^j has not established a 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 6 (Fresh and fs-fresh instance)**.** The instance Π_i^j is *fresh*,

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

The instance Π_i^j is *fs-fresh*,

1. if oracle query Reveal was not made on the instance Π_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 the instance Π_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 7. Advantage $\mathbf{Adv}_{\mathcal{P},\mathcal{A}}^{\mathrm{AKE}(-\mathrm{fs})}(\kappa)$ of an adversary $\mathcal{A}$ attacking a protocol $\mathcal{P}$ in aforementioned model without (with) forward secrecy with security parameter κ is defined by a following game:
GameAKE(-fs)$_{\mathcal{P},\mathcal{A}}$:

- $\mathcal{A}$ can ask queries to Send, Reveal, Execute (and Corrupt in case of forward secrecy) oracles multiple times.
- Test query can be asked only once by $\mathcal{A}$ and only on a fresh (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 $\mathbf{Adv}_{\mathcal{P},\mathcal{A}}^{\mathrm{AKE}(-\mathrm{fs})}(\kappa) = |2 \cdot \Pr[Succ] - 1|$.

Definition 8. 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,
- $\mathbf{Adv}_{\mathcal{P},\mathcal{A}}^{\mathrm{AKE}(-\mathrm{fs})}(\kappa) \leq \frac{Q(\kappa)}{|\mathcal{DICT}|} + \varepsilon(\kappa)$, where $\varepsilon(\kappa)$ is negligible and $Q(\kappa)$ 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.

4 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.
- Protection against the previously mentioned attacks.

We meet both these design goals by replacing the first step of the 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 LHL, our protocol works with a cyclic group G. We will use two pseudorandom hash functions $\mathcal{H}$ and $\mathcal{H}'$. 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. We assume a symmetric encryption scheme modeled as an ideal cipher $E = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ 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 message $P_i \to S : ID(P_i)||z_i^* = \mathcal{E}_{sk_i}(z_i)$. to the server
   (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 τ_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 $sk = \mathcal{H}'(K_1 \| K_2 \| \ldots \| K_n)$.

4.1 Security of MLHL protocol

Let G be a cyclic group with a 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 ε 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 the GameAKE-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 $\mathcal{A}_{\mathrm{MLHL}}$, $\mathcal{A}_{2\mathrm{P}}$, $\mathcal{A}_M$ and $\mathcal{A}_{\mathrm{DDH}}$ are denoted $t_{\mathrm{MLHL}}, t_{2\mathrm{P}}, t_M, t_{DDH}$ and κ 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 $p(\kappa)$ sessions during the 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 the protocol* MLHL *is*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}(\kappa, t_{\mathrm{MLHL}}) \leq 2\Bigg(&\frac{3(q_{\mathcal{E}}+q_{\mathcal{D}})^2}{2|G|} + \frac{3p(\kappa)\cdot n\cdot q_{\mathcal{D}}}{2^{l_k}} \\ &+ p(\kappa)\cdot n\mathbf{Adv}^{\mathrm{AKE}}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}(t_{2\mathrm{P}}, q_{execute}, q_{send}, q_{reveal}) + 2\varepsilon \\ &+ \frac{np(\kappa)^2}{2^{l_k+1}} + 5p(\kappa)\cdot n\cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t_{DDH}) + 8q_{\mathcal{E}}/2^{l_k} \\ &+ 4\mathbf{Adv}^{\mathrm{MAC-forge}}_{M,\mathcal{A}_M}(t_M)\Bigg).\end{aligned}$$

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

- $Case_1$: No Corrupt query was made during the execution of the game GameAKE-fs$_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}$. 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-fs$_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}$ (note that in this case the session key was established for instance on which Test query was made). Here are allowed Send, Execute and Reveal queries.
- $Case_3$: 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-fs$_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}$. In this case, only Execute and Reveal queries are allowed, due to preservation of the 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:

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

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. Due to space limitations, the full proof of theorem is in full version [Mi13]. The proof of the AKE security of the MLHL protocol is in Appendix A. □

4.1.1 Acknowledgement.

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

References

[BM92] 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.

[LHL04] 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.

[ABC06] 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.

[Ch06] 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.

[AFP05] 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.

[BPR00] 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.

[Kw06] 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.

[BR95] 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.

[CPS08] 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.

[KL07] Katz, Jonathan and Lindell, Yehuda, Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series),Chapman & Hall/CRC, 2007.

[KOY03] 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.

[BMP00] 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.

[Ja96] David P. Jablon, Strong Password-Only Authenticated Key Exchange, SIGCOMM Computer Communication Review 26, pp. 5–26, ACM, 1996.

[Wu98] 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.

[GL01] 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.

[KOY01] 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.

[Mi13] Marika Mitrengová, Multi-LHL protocol, Cryptology ePrint Archive, Report 2013/621, `http://eprint.iacr.org/`, 2013.

[HR11] Feng Hao and Peter Ryan, Password Authenticated Key Exchange by Juggling,Security'08 Proceedings of the 16th International conference on Security protocols, pp. 159–171, Springer, 2011.

[Ab11] Michel Abdalla1 and Celine Chevalier and Louis Granboulan and David Pointcheval, Contributory Password-Authenticated Group Key Exchange with Join Capability,CT-RSA,pp. 142–160, LNCS, Springer,2011.

A 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. If no Corrupt queries are made, it is sufficient to prove the AKE security instead of AKE-fs.

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

$$\begin{aligned}\boldsymbol{Adv}^{\mathrm{AKE-fs}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}},Case_1}(t_{\mathrm{MLHL}}) \leq 2\Bigg(& \frac{(q_{\mathcal{E}}+q_{\mathcal{D}})^2}{2|G|} + \frac{p(\kappa)\cdot n\cdot q_{\mathcal{D}}}{2^{l_k}} \\ & + p(\kappa)\cdot n\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{2P}}(t_{\mathrm{2P}}, q_{execute}, q_{send}, q_{reveal}) \\ & + \frac{np(\kappa)^2}{2^{l_k+1}} + 2p(\kappa)\cdot n\cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t_{DDH}) \\ & + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{M}(t_M) + 4q_{\mathcal{E}}/2^{l-k}\Bigg).\end{aligned}$$

Proof. We start with the simulation of the real protocol.
Game G_0:
This is a game simulating the real protocol. From the definition 7 we have:

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

Because 2P could represent arbitrary secure 2PAKE protocol, without the loss of generality we assume that the protocol has l flows of messages. By sk_i =2P(P_i, S) we denote, that the key sk_i was computed with simulation of 2P between P_i and S. When a participant awaits more than one message, we denote it as a concatenation (see definitions of Send$_3$ and Send$_4$ oracles). In this game we simulate Send oracles as described bellow (we skip the description of Execute, 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 the 2P protocol). Such Send query represents concrete type of message, which was sent.

Send$^1_1(\Pi^j_i, Msg)$
simulate first step of the 2P protocol, message Msg of the form $(ID(P_{i_1})||ID(P_{i_2})||\ldots||ID(P_{i_{n-1}})||ID(S))$ is sent to the instance Π^j_i informing that the instance Π^j_i 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 the 2P protocol.

⋮

$\text{Send}_1^l(\psi^s, Msg)$
simulate the last step of the 2P protocol,
return the last message of the 2P protocol computed according to the rules of 2P.

$\text{Send}_2^1(\Pi_i^j, Msg)$
Msg is the last message sent by the server ψ to Π_i^j in 2P,
$sk_i =$2P(P_i, S),
$x_i \xleftarrow{\$} G$,
$z_i = g^{x_i}$, $z_i^* = \mathcal{E}_{sk_i}(z_i)$
return $(ID(P_i)||z_i^*)$

$\text{Send}_2^2(\psi^s, Msg)$
Msg has the form $(ID(P_i)||Msg')$
$z_i = \mathcal{D}_{sk_i}(Msg)$,
$z_{i-1}^{**} = \mathcal{E}_{sk_{i-1}}(z_i)$,
$z_{i+1}^{**} = \mathcal{E}_{sk_{i+1}}(z_i)$
return $(ID(S)||ID(P_i)||z_{i-1}^{**})$, $(ID(S)||ID(P_i)||z_{i+1}^{**})$

$\text{Send}_3(\Pi_i^j, Msg_{i-1}||Msg_{i+1})$
Msg_{i-1} and Msg_{i+1} have the form $(ID(S)||ID(P_{i-1})||Msg'_{i-1})$ and $(ID(S)||ID(P_{i+1})||Msg'_{i+1})$
$z_{i-1} = \mathcal{D}_{sk_i}(Msg_{i-1})$, $z_{i+1} = \mathcal{D}_{sk_i}(Msg_{i+1})$,
$K_{i-1} = \mathcal{H}(z_{i-1}^{x_i})$, $K_i = \mathcal{H}(z_{i+1}^{x_i})$,
$w_i = K_{i-1} \oplus K_i$, $\tau_i = \text{Mac}_{K_i}(ID(P_i)||w_i)$
return $(ID(P_i)||w_i||\tau_i)$

$\text{Send}_4(\Pi_i^j, Msg_0||\ldots||Msg_{i-1}||Msg_{i+1}||\ldots||Msg_n)$
Msg_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 $K_{i-2} = w_{i-1} \oplus K_{i-1}, \ldots$
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 a 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 a session ID of the instance Π_i^j, α is value used in other games, sk is encryption/decryption key and $z^* = \mathcal{E}_{sk}(z)$. Moreover, we use a list $\Lambda_{2\text{P}}$ of tuples (sid, i, sk) where we store previously established session keys sk_i in the 2P protocol in session sid for the 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 a 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 a 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 a 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 a number of encryptions and $q_{\mathcal{D}}$ is a 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 $\frac{p(\kappa)\cdot n\cdot q_{\mathcal{D}}}{2^{l_k}}$, where $p(\cdot)$ is a polynomial and $q_{\mathcal{D}}$ denotes number of decryptions ($p(\kappa)\cdot n$ is number of 2P's executions).

Hence,

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

Next, we simulate gradual replacement of values sk_i by random keys in the games G_1^i. We alter the simulation of Execute and Send_2^1 queries as follows: session key sk_i established during the 2P protocol between the participant P_i and server S is replaced by a random string sk_i', while we keep these randomly chosen values in a 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 $\frac{p(\kappa)^2}{2^{l_k+1}}$, where $p(\kappa)$ specifies number of simulations of the MLHL protocol.

Game G_1^1:

In this game the session key established during the 2P protocol between participant P_1 and server S is replaced by a random string sk_1'. We store values $(sid_1^j,1,sk_1')$ in the list $\Lambda_{2\mathrm{P}}$. We show that

$$|\Pr[Succ_1^1] - \Pr[Succ_0']| \leq p(\kappa)\mathbf{Adv}^{\mathrm{AKE}}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}(t_{2\mathrm{P}},q_{execute},q_{send},q_{reveal}) + \frac{p(\kappa)^2}{2^{l_k+1}},$$

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 $\mathcal{A}_D$ that can distinguish games G'_0 and G^1_1 with probability $\varepsilon = |\Pr[\mathcal{A}_D^{G'_0} \to 1] - \Pr[\mathcal{A}_D^{G^1_1} \to 1]|$. We show that if ε is not negligible, we can construct an adversary $\mathcal{A}_{2\mathrm{P}}$ against the AKE security of the 2P protocol, which probability of success is not negligible. We define sequence distributions H^i_1, $i = 0 \dots p(\kappa)$. In the distribution H^i_1 the first i session keys established during the 2P protocol between participant P_1 and server S are replaced by a random string sk'_1. Clearly the distribution H^0_1 is equal to the game G'_0 and $H^{p(\kappa)}_1$ to G^1_1.

Adversary $\mathcal{A}_{2\mathrm{P}}$

1. $\mathcal{A}_{2\mathrm{P}}$ selects an index j at random from $\{1 \dots p(\kappa) - 1\}$ and a bit $b \xleftarrow{\$} \{0, 1\}$. Then $\mathcal{A}_{2\mathrm{P}}$ runs distinguisher $\mathcal{A}_D$ and responds to his oracle queries (described later). We assume that $\mathcal{A}_{2\mathrm{P}}$ is able to identify, which queries asked by $\mathcal{A}_D$ belong to the 2P protocol ($\mathrm{Send}^1_1, \dots, \mathrm{Send}^l_1$) and which belong to the rest of the protocol MLHL ($\mathrm{Send}^1_2, \dots, \mathrm{Send}_4$). $\mathcal{A}_{2\mathrm{P}}$ will simulate oracle queries of $\mathcal{A}_D$ as follows:
   - $\mathrm{Send}(\Pi^l_1, Msg)$ in 2P, $l < j$: $\mathcal{A}_{2\mathrm{P}}$ replies with the response of his $\mathrm{Send}(\Pi^l_1, Msg)$ 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 sid^l_1.
   - $\mathrm{Send}(\Pi^j_1, Msg)$ in 2P: $\mathcal{A}_{2\mathrm{P}}$ replies with the response of his $\mathrm{Send}(\Pi^j_1, Msg)$ oracle. If this query leads to establishment of a session key in 2P, then $\mathcal{A}_{2\mathrm{P}}$ asks $\mathrm{Test}(\Pi^j_1)$ query and a result is used as a session key sk_1 between P_1 and S in the MLHL protocol with the session identifier sid^j_1.
   - $\mathrm{Send}(\Pi^l_i, Msg)$ in 2P, $i \neq 1 \wedge l \in \{1, \cdots p(\kappa)\}$ or $i = 1 \wedge l > j$: $\mathcal{A}_{2\mathrm{P}}$ replies with the response of his $\mathrm{Send}(\Pi^j_i, Msg)$ oracle. If this query leads to establishment of a session key in 2P, then $\mathcal{A}_{2\mathrm{P}}$ asks $\mathrm{Reveal}(\Pi^l_i)$ query and the returned result is used as a session key sk_i between P_i and S in the session sid^j_i.
   - $\mathrm{Send}(\Psi^s, Msg)$ in 2P: similar as $\mathrm{Send}(\Pi^j_i, Msg)$
   - $\mathrm{Send}(\Pi^j_i, Msg)$ query outside 2P: $\mathcal{A}_{2\mathrm{P}}$ answers with the result of simulation of sending the message Msg 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).
   - $\mathrm{Send}(\Psi^s, Msg)$ query outside 2P: similar to $\mathrm{Send}(\Pi^j_i, Msg)$ outside 2P.
   - $\mathrm{Execute}(P_1, P_2, \dots, P_n, S)$: similar to combination of the Send queries.
   - $\mathrm{Reveal}(\Pi^j_i)$: $\mathcal{A}_{2\mathrm{P}}$ answers under the rules of Reveal query in the security model (he returns a real session key sk, if Π^j_i has the key established during the simulation)
   - $\mathrm{Test}(\Pi^j_i)$: if a 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 $b \leftarrow \mathcal{D}$

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

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

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

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

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 the distinguisher $\mathcal{A}_D$ is as in the distribution H_1^J. That is,

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

Then, we have

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

In the end we have

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

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

$$\begin{aligned}|\Pr[Succ_1^1] - \Pr[Succ_0']| &= |\Pr_{view \leftarrow H_1^{p(\kappa)}}[\mathcal{A}_D(view) = 1] - \Pr_{view \leftarrow H_1^0}[\mathcal{A}_D(view) = 1]| \\ &\leq p(\kappa)\mathbf{Adv}^{\mathrm{AKE}}_{2\mathrm{P},\mathcal{A}_{2\mathrm{P}}}(t_{2\mathrm{P}}, q_{execute}, q_{send}, q_{reveal}) + \frac{p(\kappa)^2}{2^{l_k+1}}.\end{aligned}$$

Games $G_1^2, \ldots, G_1^n$ are defined similarly. The similar reasoning of existence of a distinguisher between games G_1^i and G_1^{i+1} works. When we sum all inequalities on the left side and on the right side,

$$|\Pr[Succ_1^n] - \Pr[Succ_0']| \leq n \cdot p(\kappa)\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{2P},\mathcal{A}_{\mathrm{2P}}}(t_{\mathrm{2P}}, q_{execute}, q_{send}, q_{reveal}) + \frac{np(\kappa)^2}{2^{l_k+1}}.$$

In this part we simulate gradual replacement of values K_i by random values in the games $G_2^i, i = 1 \ldots n$. 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: We simulate everything like in the previous game in this game, however the value K_1 is replaced by a random value during Execute queries. We show that

$$|\Pr[Succ_2^1] - \Pr[Succ_1^n]| \leq p(\kappa)\mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t_{DDH}).$$

To prove this inequality, suppose that there exist a distinguisher $\mathcal{A}_D$ 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 a distribution H_2^i, $i \in \{0, 1, \ldots p(k)\}$. In the distribution H_2^i the values K_1 for instances Π_1^j, $j \leq i$ are chosen randomly and the values K_1 for instances Π_1^j, $j > i$ are computed as in the previous game. We assume that distinguisher $\mathcal{A}_D$ constructs $p(\kappa)$ sessions for some polynomial $p(\cdot)$ during simulation.

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

1. $\mathcal{A}_{\mathrm{DDH}}$ chooses a random bit b and an index j.
2. $\mathcal{A}_{\mathrm{DDH}}$ answers oracle queries of the distinguisher $\mathcal{A}_D$ as follows:
   - Send, Reveal and Test queries are answered as in the previous game, Test queries are answered with the use of the bit b (note, that the adversary knows established session keys, because he simulated the execution).
   - Execute$(P_1, \ldots, P_n)$ queries are simulated in the following way:
     If instance of P_1 has form Π_1^l, where $l = j$ then simulation of Execute query for instances of participants $P_3, \ldots, P_n$ in the same session does not change. The protocol 2P between P_1, P_2 and S is simulated as in the previous game, after this simulation $\mathcal{A}_{\mathrm{DDH}}$ knows values sk_1, sk_2 – he has chosen them randomly. Then he simulates that P_1 sends a message $(ID(P_1)||\mathcal{E}_{sk_1}(u))$ and P_2 sends a 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 Π_1^l, where $l < j$ then $\mathcal{A}_{\mathrm{DDH}}$ starts to simulate 2P between participants $P_1, \ldots, P_n$ and S as in the previous game. After simulation of the 2P protocol he chooses randomly keys sk_i', $i = 1 \ldots n$ and simulates the rest of the MLHL as in the previous game however, a computed value K_1 in each session is replaced by a random value.

– If instance of P_1 has form Π_1^l, where $l > j$ then $\mathcal{A}_{\mathrm{DDH}}$ starts to simulate 2P between participants $P_1, \ldots, P_n$ and S as in the previous game. After simulation of the 2P protocol he continues with simulation of the rest of the MLHL as in the previous game.

3. $\mathcal{A}_{DDH}$ returns a $b \leftarrow \mathcal{A}_D$

If (u, v, w) from adversary's input is a DDH triple and the index $j = 0$, the view of the distinguisher $\mathcal{A}_D$ is the same as in the game G_1^n (H_2^0). If (u, v, w) is not a DDH triple and the index $j = p(k)$, the view of $\mathcal{A}_D$ 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 $\frac{1}{p(\kappa)}$ of the advantage of $\mathcal{A}_D$ (we skip the detailed reasoning).

$$|\Pr[Succ_2^1] - \Pr[Succ_1^n]| = p(\kappa)\mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t_{DDH})$$

The games $G_2^2, \ldots, G_2^n$ are defined similarly. When we sum inequalities, we have

$$|\Pr[Succ_2^n] - \Pr[Succ_1^n]| = n \cdot p(\kappa) \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t_{DDH}).$$

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 the decryption oracle (defined in the 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 $(\alpha_1, ..., \alpha_n)$, $\alpha_i \xleftarrow{\$} Z_q^*$, vectors $(\zeta_1^{\alpha_1}, ..., \zeta_n^{\alpha_n}, \gamma_1^{\alpha_1\alpha_2}, ..., \gamma_n^{\alpha_n\alpha_1})$ have 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 the session ID), we choose $\alpha_i \xleftarrow{\$} Z_q^*$ randomly 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 α_i specifies, how we applied random self-reducibility of PDDH problem on instance generated by the challenger. Exponent α_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 Send_2^1, Send_2^2 and Send_3 queries. First, encryption of messages in the second step of the protocol is changed during simulation of Send_2^1. The instance Π_i^j chooses $z_i^* \xleftarrow{\$} G$ randomly and computes $z_i = \mathcal{D}_{sk_i}(z_i^*)$ as in the previous game. Then Π_i^j sends a message $(ID(P_i)||z_i^*)$. Therefore Send_2^1 queries in the second step of the MLHL lead to adding of α_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 α_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 the Game G_0', point 3).

When the server accepts the message $(ID(P_i)||z_i^*)$ during simulation of 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 α_i. We can continue with the simulation of encryption described bellow.
- z_i^* is response of the encryption oracle $\mathcal{E}_{sk_i}$, while sk_i is a correct key of Π_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 the adversary without asking the encryption oracle. In this situation the adversary does not know the password and therefore he could not compute messages in the way they go through the control step. The 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} a message $(ID(S)||ID(P_i)||z_i^{**})$. Similar for P_{i-1}. Next we continue in simulation as in previous games, however, the adversary does not know any of values K_i, therefore he could not manipulate other messages in the way they go through the verification step of the 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 the simulation described above by the instance Π_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 the 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 the 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 a request for resending the value z_i to the instance Π_{i+1}^j (sent with the instance Π_{i+2}^j) happens, 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 the instance Π_{i+1}^j in the second round.
- $\mathcal{E}_{sk_{i-1}}(z_i)$ – similar to the previous case.

The simulation of Send$_3$, when Π_i^j receives a 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 the simulation as described bellow.
- one or both z_{i-1}^* and z_{i+1}^* is/are the answer from query on the encryption oracle $\mathcal{E}_{sk_i}$, while sk_i is correct key of the instance Π_i^j in the session j (thus adversary guessed a password and used its 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 the adversary without asking for the Encryption oracle, in this situation the adversary does not know the password and therefore he could not compute messages in the way they go through the verification step of the MAC scheme, unless he breaks it with probability $\mathbf{Adv}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}^{\mathrm{MAC-forge}}(t_M)$, which is negligible.

If 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 a 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.

$$|\Pr[Succ_5] - \Pr[Succ_4]| \leq 4q_{\mathcal{E}}/2^{l_k} + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}(t_M)$$

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

$$|\Pr[Succ_6] - \Pr[Succ_5]| \leq p(\kappa) \cdot \mathbf{Adv}^{\mathrm{PDDH}_n}_{G,\mathcal{A}_{\mathrm{PDDH}}}(t_{DDH}),$$

where $p(\kappa)$ 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 the view of the adversary in this two games. In the game G_6 values K_i are chosen randomly, 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 the adversary's success in this game is $\Pr[Succ_7] = \frac{1}{2}$, because the session key is randomly chosen and independent from the previous messages. When we sum all (in)equalities of games, we have:

$$\begin{aligned}|\Pr[Succ_7] - \Pr[Succ_0]| \leq{}& \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(\kappa) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}(t_M) \\ &+ p(\kappa) \cdot n\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{2P},\mathcal{A}_{\mathrm{2P}}}(t_{\mathrm{2P}}, q_{execute}, q_{send}, q_{reveal}) \\ &+ \frac{np(\kappa)^2}{2^{l_k+1}} + p(\kappa) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t_{DDH}) + 4q_{\mathcal{E}}/2^{l-k} \\ &+ p(\kappa) \cdot \mathbf{Adv}^{\mathrm{PDDH}_n}_{G,\mathcal{A}_{\mathrm{PDDH}}}(t_{DDH})\end{aligned}$$

$$\begin{aligned}\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{MLHL},\mathcal{A}_{\mathrm{MLHL}}}(\mathcal{A}) \leq 2\Bigg(& \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(\kappa) \cdot n \cdot q_{\mathcal{D}}}{2^{l_k}} + 2\mathbf{Adv}^{\mathrm{MAC-forge}}_{\mathrm{M},\mathcal{A}_{\mathrm{M}}}(t_M) \\ &+ p(\kappa) \cdot n\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{2P},\mathcal{A}_{\mathrm{2P}}}(t_{\mathrm{2P}}, q_{execute}, q_{send}, q_{reveal}) \\ &+ \frac{np(\kappa)^2}{2^{l_k+1}} + 2p(\kappa) \cdot n \cdot \mathbf{Adv}^{\mathrm{DDH}}_{G,\mathcal{A}_{\mathrm{DDH}}}(t_{DDH}) + 4q_{\mathcal{E}}/2^{l-k}\Bigg).\end{aligned}$$

□