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 keys 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 $DICT$ 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-Hellman 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 $f : Y_1 \rightarrow Y_2$ uniformly chosen from the set $\text{Func}(Y_1, Y_2)$ 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 $\bot$ represents undefined value.

A symmetric encryption scheme is denoted as $E = (G, E, D)$ and message authentication code scheme (MAC) is denoted as $M = (\text{Gen}, \text{Mac}, \text{Vrf})$. A tag $\tau$ is computed as $\tau = \text{Mac}_k(\text{Msg})$ for message $\text{Msg}$ 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 $C \cup S$, where $C = \{P_1, P_2, \ldots P_n\}$ is set of clients and $S$ is set of servers. For simplicity, we assume that $|S| = 1$. Each client $P_i \in C$ has a password $pw_i$ called long-lived key (LL-key) and server $S$ has a vector of clients passwords $\langle pw_{S,P_i} \rangle_{P_i \in C}$ ($pw_i = pw_{S,P_i}$ for all $P_i \in C$ in symmetric case, otherwise they are different in asymmetric case). The $j$-th instance of participant $P_i$ is denoted as $\Pi^j_i$ 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
$P_1, P_2, \ldots, P_h$ is denoted as $Grp_{i_1, i_2, \ldots, i_h}$.

**Definition 1.** [BR95] 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.

**Definition 2.** An adversary is a probabilistic polynomial-time Turing machine with oracle access to several other Turing machines. Running time of an adversary $A$ is the length of description of $A$ plus the worst case running time of $A$.

Let $CON$ be a cryptographic construction (algorithm), $A$ be an adversary and $xxx$ be any problem on $CON$ (such as collision resistance of hash function, or discrete logarithm in a group $G$). $\text{Adv}^\text{xxx}_{CON, A}$ is a measure of adversary’s advantage defined as a probability, that $A$ 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 $\text{Adv}^\text{xxx}_{CON, A}(\kappa_1, \kappa_2, \cdots, \kappa_n)$.

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 $\text{Chall}^\beta(\cdot)$ 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 $A_D$ be an adversary (distinguisher). Two distributions are defined:

\[
PDDH_n^* = \{(g^{x_1}, g^{x_2}, \ldots, g^{x_n}, g^{x_1x_2}, g^{x_2x_3}, \ldots, g^{x_nx_1}) | x_1, x_2, \ldots, x_n \leftarrow Z_q^* \} \text{ and} \]

\[
PDDH_n^\$ = \{(g^{x_1}, g^{x_2}, \ldots, g^{x_n}, g^{y_1}, g^{y_2}, \ldots, g^{y_n}) | x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n \leftarrow Z_q^* \}, \]

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 $A_D$ on PDDH$_n$ problem in time $t_{DDH}$ is negligible. This advantage is denoted as $\text{Adv}^\text{PDDH}_n_{G,A_D}(t_{DDH})$ and computed as:

\[
\text{Adv}^\text{PDDH}_n_{G,A_D}(t_{DDH}) = |\Pr[A_D(PDDH_n^*) \rightarrow 1] - \Pr[A_D(PDDH_n^\$) \rightarrow 1]|.
\]

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

\[
\text{Adv}^\text{DDH}_{G,A_D}(t_{DDH}) \leq \text{Adv}^\text{PDDH}_n_{G,A_D}(t_{DDH}) \leq n \cdot \text{Adv}^\text{DDH}_{G,A_D}(t_{DDH})
\]

$\text{Chall}^\beta(I)$ is an algorithm that on an input $I$ outputs vectors from the distribution $PDDH_n^*$, if the bit $\beta = 0$, otherwise it outputs vectors from the distribution $PDDH_n^\$.$ 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 $\Pi_i^j$ is denoted as $sid_i^j$. For the server instance $\Psi^s$ is the session identifier 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 5. A partner identifier $pid_i^j$ for the 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 $\Pi_k^l \in pid_i^j$

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($\Pi_i^j, Msg$) – sends the message $Msg$ to the instance $\Pi_i^j$ in the 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($\Psi^s, Msg$) – similarly to the Send($\Pi_i^j, Msg$). This oracle query sends the message $Msg$ to the instance of the server $\Psi^s$ in the session $sid^s$ and returns a reply of $\Psi^s$.
- 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($\Pi_i^j$) – 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 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($\Pi_i^j$) – This query can be used only on a fresh/fs-fresh instance (see Def. 6). First a random bit $b \leftarrow \{0, 1\}$ is chosen. If instance $\Pi_i^j$ 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 $sk' \leftarrow \{0, 1\}^{sk}$ is returned.
Definition 6 (Fresh and fs-fresh instance). The 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.

The 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 the 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 7. Advantage $\text{Adv}^{\text{AKE}(-fs)}_{\mathcal{P}, \mathcal{A}}(\kappa)$ of an adversary $\mathcal{A}$ attacking a protocol $\mathcal{P}$ in aforementioned model without (with) forward secrecy with security parameter $\kappa$ is defined by a following game:

Game$\text{AKE}(-fs)_{\mathcal{P}, \mathcal{A}}$:

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

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,
- $\text{Adv}^{\text{AKE}(-fs)}_{\mathcal{P}, \mathcal{A}}(\kappa) \leq \frac{Q(\kappa)}{|DICT|} + \varepsilon(\kappa)$, 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). $DICT$ 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 $E = (G, \mathcal{E}, \mathcal{D})$ and an existentially unforgeable under an adaptive chosen-message attack secure message authentication scheme $M = (\text{Gen}, \text{Mac}, \text{Vrf})$.

**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 $P_i \rightarrow S: ID(P_i)||z_i^* = \mathcal{E}_{sk_i}(z_i)$. to the server
   (b) Server decrypts $z_i^*$ and sends following messages to the participants $P_{i-1}$ and $P_{i+1}$:
      
      $S \rightarrow P_{i-1}: ID(S)||ID(P_i)||\mathcal{E}_{sk_{i-1}}(z_i)$
      $S \rightarrow P_{i+1}: ID(S)||ID(P_i)||\mathcal{E}_{sk_{i+1}}(z_i)$
   (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 $\tau_i = \text{Mac}_{K_i}(ID(P_i)||w_i)$ 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 $K_j = \mathcal{H}(g^{x_j-x_i})$ 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 $K_{i-2} = w_{i-1} \oplus K_{i-1}, \text{Vrf}_{K_{i-1}}(ID(P_{i-1})||w_{i-1}, \tau_{i-1})$ and ends with $K_i = w_{i+1} \oplus K_{i+1}, \text{Vrf}_{K_{i+1}}(ID(P_{i+1})||w_{i+1}, \tau_{i+1})$. 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 $\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^{l_H}$ and $\mathcal{H}' : \{0, 1\}^* \rightarrow \{0, 1\}^{l_H}$.
\{0,1\}^l_{\kappa'}$. Let $2P$ be an arbitrary secure 2PAKE protocol with length of the session key $l_k$, let $E = (G, \mathcal{E}, D)$ be symmetric encryption scheme defined as $\mathcal{E} : G \times \{0,1\}^{l_k} \rightarrow G$, $D : G \times \{0,1\}^{l_k} \rightarrow G$ and modeled as an ideal cipher. Let $M = (\text{Gen, Mac, Vrf})$ be an existentially unforgeable under adaptive chosen-message attack secure message authentication scheme. Symbol $\varepsilon$ denotes a negligible function, $q_E$ number of encryption queries, $q_D$ number of decryption queries, $q_{\text{send}}, q_{\text{execute}}, q_{\text{reveal}}$ is number of Send, Execute, Reveal queries the attacker makes in underlying 2P protocol during the GameAKE-fsMLHL,\textsubscript{A\textsubscript{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 $A_X$ denotes adversary attacking construction $X$ on its security property. Running times of adversaries $A_{\text{MLHL}}, A_{2P}, A_M$ and $A_{\text{DDH}}$ are denoted $t_{\text{MLHL}}, t_{2P}, t_M, t_{\text{DDH}}$ and $\kappa$ 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 $A_{\text{MLHL}}$ establishes $p(\kappa)$ sessions during the GameAKE-fsMLHL,\textsubscript{A\textsubscript{MLHL}} between $n$ participants for some polynomial $p(\cdot)$. Then the advantage of the adversary $A_{\text{MLHL}}$ in attacking the protocol MLHL is

$$\text{Adv}_{\text{MLHL},A_{\text{MLHL}}}^{\text{AKE-fs}}(\kappa, t_{\text{MLHL}}) \leq 2 \left( \frac{3(q_E + q_D)^2}{2|G|} + \frac{3p(\kappa) \cdot n \cdot q_D}{2^{l_k}} \right) + p(\kappa) \cdot n \cdot \text{Adv}_{2P, A_{2P}}^{\text{AKE}}(t_{2P}, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}}) + 2\varepsilon$$

$$+ \frac{np(\kappa)^2}{2^{l_k+1}} + 5p(\kappa) \cdot n \cdot \text{Adv}_{G, A_{\text{DDH}}}^{\text{DDH}}(t_{\text{DDH}}) + 8q_E/2^{l_k}$$

$$+ 4\text{Adv}_{M, A_M}^{\text{MAC-forg}(t_M)}.$$

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-fsMLHL,\textsubscript{A\textsubscript{MLHL}}:

- **Case 1:** No Corrupt query was made during the execution of the game GameAKE-fsMLHL,\textsubscript{A\textsubscript{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-fsMLHL,\textsubscript{A\textsubscript{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-fsMLHL,\textsubscript{A\textsubscript{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:

\[
\text{Adv}^{\text{AKE-fs}}_{\text{MLHL}, A_{\text{MLHL}}}(t_{\text{MLHL}}) = \text{Adv}^{\text{AKE-fs}}_{\text{MLHL}, A_{\text{MLHL}}, \text{Case}_1}(t_{\text{MLHL}}) + \text{Adv}^{\text{AKE-fs}}_{\text{MLHL}, A_{\text{MLHL}}, \text{Case}_2}(t_{\text{MLHL}}) + \text{Adv}^{\text{AKE-fs}}_{\text{MLHL}, A_{\text{MLHL}}, \text{Case}_3}(t_{\text{MLHL}}).
\]

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 \( \text{Succ}_i \) 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 \( \text{Send} \) queries, we assume that the number of users is \( n \) too.

\[\text{Proof.}\] 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.

\[\square\]

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


Multi-LHL protocol


[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.


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:

\[
\text{Adv}_{\text{MLHL}, \text{A}_{\text{MLHL}}, \text{Case}_1}^{\text{AKE-}fs} (t) \leq 2 \left( \frac{(q_E + q_D)^2}{2|G|} + \frac{p(\kappa) \cdot n \cdot q_D}{2^{l_k}} \right)
\]
\[
+ p(\kappa) \cdot n \text{Adv}_{2\text{P}}^{\text{AKE}} (t_{2\text{P}}, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}})
\]
\[
+ \frac{np(\kappa)^2}{2^{l_k+1}} + 2p(\kappa) \cdot n \cdot \text{Adv}_{G, \text{ADDI}}^{\text{DDH}} (t_{DDH})
\]
\[
+ 2 \text{Adv}_{M}^{\text{MAC-forge}} (t_M) + 4q_E / 2^{l_k-1} \right).
\]

Proof. We start with the simulation of the real protocol.

Game G0:
This is a game simulating the real protocol. From the definition 7 we have:

\[
\text{Adv}_{\text{MLHL}, \text{A}_{\text{MLHL}}, \text{Case}_1}^{\text{AKE}} (t) = 2 \Pr[Succ_0] - 1.
\]

Because 2P could represent arbitrary secure 2PAKE protocol, without the loss of generality we assume that the protocol has \(l\) flows of messages. By \(sk_i = 2\text{P}(P_i, S)\) 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 below (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.

\[
\text{Send}_i^1(\Pi_i^j, Msg)
\]

- 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 \(\Pi_i^j\) 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 the 2P protocol.
Send\textsuperscript{1}(ψ\textsuperscript{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.

Send\textsuperscript{2}(Π\textsuperscript{j}, Msg)

Msg is the last message sent by the server ψ to Π\textsuperscript{j} in 2P,
sk\textsubscript{i} = 2P(P\textsubscript{i}, S),
x\textsubscript{i} ← G,
z\textsubscript{i} = g\textsuperscript{x\textsubscript{i}}, z\textsuperscript{*\textsubscript{i}} = E\textsubscript{sk\textsubscript{i}}(z\textsubscript{i})
return (ID(P\textsubscript{i})||z\textsuperscript{*\textsubscript{i}})

Send\textsuperscript{3}(ψ\textsuperscript{s}, Msg)

Msg has the form (ID(P\textsubscript{i})||Msg′)
z\textsubscript{i} = D\textsubscript{sk\textsubscript{i}}(Msg),
z\textsuperscript{*\textsubscript{i}} = E\textsubscript{sk\textsubscript{i}}(z\textsubscript{i}),
z\textsuperscript{**\textsubscript{i}} = E\textsubscript{sk\textsubscript{i+1}}(z\textsubscript{i})
return (ID(S)||ID(P\textsubscript{i})||z\textsuperscript{**\textsubscript{i}}), (ID(S)||ID(P\textsubscript{i})||z\textsuperscript{**\textsubscript{i+1}})

Send\textsuperscript{3}(Π\textsuperscript{j}, Msg\textsubscript{i-1}||Msg\textsubscript{i+1})

Msg\textsubscript{i-1} and Msg\textsubscript{i+1} have the form (ID(S)||ID(P\textsubscript{i-1})||Msg\textsubscript{i-1}′) and (ID(S)||ID(P\textsubscript{i+1})||Msg\textsubscript{i+1}′)

z\textsubscript{i-1} = D\textsubscript{sk\textsubscript{i}}(Msg\textsubscript{i-1}), z\textsubscript{i+1} = D\textsubscript{sk\textsubscript{i}}(Msg\textsubscript{i+1}),
K\textsubscript{i-1} = H(z\textsuperscript{**\textsubscript{i-1}}, K\textsubscript{i} = H(z\textsuperscript{**\textsubscript{i+1}}),
w\textsubscript{i} = K\textsubscript{i-1} + K\textsubscript{i}, τ\textsubscript{i} = Mac\textsubscript{K\textsubscript{i}}(ID(P\textsubscript{i})||w\textsubscript{i})
return (ID(P\textsubscript{i})||w\textsubscript{i}||τ\textsubscript{i})

Send\textsuperscript{4}(Π\textsuperscript{j}, Msg\textsubscript{0}||...||Msg\textsubscript{i-1}||Msg\textsubscript{i+1}||...||Msg\textsubscript{n})

Msg\textsubscript{j} has the form (ID(P\textsubscript{j})||w\textsubscript{j}||τ\textsubscript{j}),
j ∈ \{0, ..., i−1, i+1, ..., n\},
if Vrf\textsubscript{K\textsubscript{i-1}}(ID(P\textsubscript{i-1})||w\textsubscript{i-1}||τ\textsubscript{i-1}) = 1
then K\textsubscript{i-2} = w\textsubscript{i-1} + K\textsubscript{i-1},...
if Vrf\textsubscript{K\textsubscript{i+1}}(ID(P\textsubscript{i+1})||w\textsubscript{i+1}||τ\textsubscript{i+1}) = 1
then K\textsubscript{i} = w\textsubscript{i+1} + K\textsubscript{i+1},
sk = H′(H(K\textsubscript{1})||...||H(K\textsubscript{n}))
return "accept"
else if any of MAC verifications fails, return "terminated"

Game G\textsubscript{0}:
In this game we simulate encryption and decryption oracles. We work with a list Λ\textsubscript{enc} of tuples (type, sid\textsubscript{i}, i, α, sk\textsubscript{i}, z, z\textsuperscript{*}), where we store previous answers of encryption/decryption queries. Type takes values enc/dec, sid\textsubscript{i} is a session ID of the instance Π\textsubscript{j}, α is value used in other games, sk\textsubscript{i} is encryption/decryption key and z\textsuperscript{*} = E\textsubscript{sk\textsubscript{i}}(z). Moreover, we use a list Λ\textsubscript{2P} of tuples (sid, i, sk) where we store previously established session keys sk\textsubscript{i} in the 2P protocol in session sid for the participant P\textsubscript{i}. We simulate encryption and decryption as follows:
• \( \mathcal{E}_{sk}(z) \) if \( (\cdot, \ldots, \cdot, sk, z, z^*) \in \Lambda_G \), we return \( z^* \) otherwise we choose \( z^* \in G \), if \( (\cdot, \ldots, \cdot, sk, z^*) \in \Lambda_G \), we stop the simulation and the adversary wins (because such situation represents a collision). Otherwise we add a record \((\text{enc}, \bot, \bot, \bot, sk, z, z^*)\) to \( \Lambda_G \) and return \( z^* \).

• \( \mathcal{D}_{sk}(z^*) \) if \( (\cdot, \ldots, \cdot, sk, z, z^*) \in \Lambda_G \), we return \( z \) otherwise
  - if \( (sid_i^i, i, sk) \in \Lambda_{2P} \), we choose \( z \in \mathcal{G} \), if \( (\cdot, \cdot, \cdot, sk, z, \cdot) \in \Lambda_G \), we stop the simulation and the adversary wins. Otherwise we return \( z \) and add a record \((\text{dec}, sid_i^i, i, \bot, sk, z, z^*)\) to \( \Lambda_G \).
  - if \( (sid_i^i, i, sk) \notin \Lambda_{2P} \), we choose \( z \in \mathcal{G} \), if \( (\cdot, \cdot, \cdot, sk, z, \cdot) \in \Lambda_G \), we stop the simulation and the adversary wins. Otherwise we return \( z \) and add a record \((\text{dec}, \bot, \bot, \bot, sk, z, z^*)\) to \( \Lambda_G \).

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 \( D \) and then returned as a result of the 2P protocol in the first step of the protocol MLHL. This event occurs with probability \( \frac{p(\cdot)n \cdot q_{\mathcal{D}}}{2lk} \), where \( p(\cdot) \) is a polynomial and \( q_{\mathcal{D}} \) denotes number of decryptions (\( p(\cdot) \cdot n \) is number of 2P’s executions).

Hence,

\[
|\Pr[\text{Succ}_0^1] - \Pr[\text{Succ}_0]| \leq \frac{(q_{\mathcal{E}} + q_{\mathcal{D}})^2}{2|G|} + \frac{p(\cdot)n \cdot q_{\mathcal{D}}}{2lk}.
\]

Next, we simulate gradual replacement of values \( sk_i \) by random keys in the games \( G_i \). We alter the simulation of Execute and Send 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 \( \Lambda_{2P} \) in the format \((sid_i^i, i, sk'_i)\). 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 \( \frac{p(\cdot)^2}{2lk+1} \), where \( p(\cdot) \) 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 \( (sid_1^1, 1, sk'_1) \) in the list \( \Lambda_{2P} \). We show that

\[
|\Pr[\text{Succ}_1^1] - \Pr[\text{Succ}_0^1]| \leq p(\cdot)\text{Adv}_{2P,\text{A2P}}^{\text{A2P}}(t_{2P}, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}}) + \frac{p(\cdot)^2}{2lk+1},
\]

where \( q_{\text{send}}, q_{\text{execute}}, q_{\text{reveal}} \) 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 $A_D$ that can distinguish games $G'_0$ and $G'_1$ with probability $\varepsilon = |\Pr[A_D^{G'_0} \rightarrow 1] - \Pr[A_D^{G'_1} \rightarrow 1]|$. We show that if $\varepsilon$ is not negligible, we can construct an adversary $A_{2P}$ against the AKE security of the 2P protocol, which probability of success is not negligible. We define sequence distributions $H_i^1$, $i = 0 \ldots p(\kappa)$. In the distribution $H_1^1$, the first $i$ session keys established during the 2P protocol between participant $P_1$ and server $S$ are replaced by a random string $sk'_i$. Clearly the distribution $H_1^0$ is equal to the game $G'_0$ and $H_1^{p(\kappa)}$ to $G'_1$.

**Adversary $A_{2P}$**

1. $A_{2P}$ selects an index $j$ at random from $\{1 \ldots p(\kappa) - 1\}$ and a bit $b \leftarrow \{0, 1\}$. Then $A_{2P}$ runs distinguisher $A_D$ and responds to his oracle queries (described later). We assume that $A_{2P}$ is able to identify, which queries asked by $A_D$ belong to the 2P protocol (Send$^1_1, \ldots, \text{Send}^1_4$) and which belong to the rest of the protocol MLHL (Send$^1_1, \ldots, \text{Send}^4_1$). $A_{2P}$ will simulate oracle queries of $A_D$ as follows:

- **Send($\Pi^l_1, Msg$) in 2P, $l < j$:** $A_{2P}$ replies with the response of his Send($\Pi^l_1, Msg$) oracle. If this query leads to establishment of a session key in 2P, then $A_{2P}$ selects a random key $sk'_1$ and uses it as a session key $sk_1$ between $P_1$ and $S$ in the session $sid^l_1$.
- **Send($\Pi^l_1, Msg$) in 2P: $A_{2P}$ replies with the response of his Send($\Pi^l_1, Msg$) oracle. If this query leads to establishment of a session key in 2P, then $A_{2P}$ asks Test($\Pi^l_1$) 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 $sid^l_1$.
- **Send($\Pi^l_1, Msg$) in 2P, $i \neq 1 \land l \in \{1, \ldots, p(\kappa)\}$ or $i = 1 \land l > j$: $A_{2P}$ replies with the response of his Send($\Pi^l_1, Msg$) oracle. If this query leads to establishment of a session key in 2P, then $A_{2P}$ asks Reveal($\Pi^l_1$) query and the returned result is used as a session key $sk_i$ between $P_i$ and $S$ in the session $sid^l_i$.
- **Send($\Psi^s, Msg$) in 2P: similar as Send($\Pi^l_1, Msg$)**
- **Send($\Pi^l_1, Msg$) query outside 2P: $A_{2P}$ 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 $sk_i, i \in \{1, 2, \ldots, n\}$ (which were obtained as a response of his Reveal or Test oracle or by a random choice).
- **Send($\Psi^s, Msg$) query outside 2P: similar to Send($\Pi^l_1, Msg$) outside 2P.**
- **Execute($P_1, P_2, \ldots, P_n, S$): similar to combination of the Send queries.**
- **Reveal($\Pi^l_1$): $A_{2P}$ answers under the rules of Reveal query in the security model (he returns a real session key $sk$, if $\Pi^l_1$ has the key established during the simulation)**
- **Test($\Pi^l_1$): if a randomly chosen bit $b = 0$, $A_{2P}$ returns the real session key $sk$ (computed during the simulation of Send or Execute queries), otherwise he returns a randomly chosen key $sk'$.**
2. $A_{2P}$ returns $b \leftarrow D$

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

$$\Pr_{sk_1 \leftarrow 2P(P_1, S)} [A_{2P}(sk_1) = 1|j = J] = \Pr_{\text{view} \leftarrow H_1^{j-1}} [A_D(\text{view}) = 1].$$

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

$$\Pr_{sk_1 \leftarrow 2P(P_1, S)} [A_{2P}(sk_1) = 1] = \frac{1}{p(\kappa)} \sum_{j=1}^{p(\kappa)} \Pr_{sk_1 \leftarrow 2P(P_1, S)} [A_{2P}(sk_1) = 1|j = J]$$

$$= \frac{1}{p(\kappa)} \sum_{j=1}^{p(\kappa)} \Pr_{\text{view} \leftarrow H_1^{j-1}} [A_D(\text{view}) = 1].$$

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

$$\Pr_{sk_1 \leftarrow \{0,1\}^k} [A_{2P}(sk_1) = 1|j = J] = \Pr_{\text{view} \leftarrow H_1^J} [A_D(\text{view}) = 1].$$

Then, we have

$$\Pr_{sk_1 \leftarrow \{0,1\}^k} [A_{2P}(sk_1) = 1] = \frac{1}{p(\kappa)} \sum_{j=1}^{p(\kappa)} \Pr_{sk_1 \leftarrow \{0,1\}^k} [A_{2P}(sk_1) = 1|j = J]$$

$$= \frac{1}{p(\kappa)} \sum_{j=1}^{p(\kappa)} \Pr_{\text{view} \leftarrow H_1^J} [A_D(\text{view}) = 1].$$

In the end we have

$$\left| \Pr_{sk_1 \leftarrow \{0,1\}^k} [A_{2P}(sk_1) = 1] - \Pr_{sk_1 \leftarrow 2P(P_1, S)} [A_{2P}(sk_1) = 1] \right|$$

$$= \frac{1}{p(\kappa)} \left| \sum_{j=1}^{p(\kappa)} \Pr_{\text{view} \leftarrow H_1^J} [A_D(\text{view}) = 1] - \sum_{J=0}^{p(\kappa)-1} \Pr_{\text{view} \leftarrow H_1^J} [A_D(\text{view}) = 1] \right|$$

$$= \frac{1}{p(\kappa)} \left| \Pr_{\text{view} \leftarrow H_1^{p(\kappa)}} [A_D(\text{view}) = 1] - \Pr_{\text{view} \leftarrow H_1^0} [A_D(\text{view}) = 1] \right| = \frac{\varepsilon}{p(\kappa)}.$$

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

$$\left| \Pr[\text{Succ}_1] - \Pr[\text{Succ}_0] \right| = \left| \Pr_{\text{view} \leftarrow H_1^{p(\kappa)}} [A_D(\text{view}) = 1] - \Pr_{\text{view} \leftarrow H_1^0} [A_D(\text{view}) = 1] \right|$$

$$\leq p(\kappa) \text{Adv}_{2P,A_{2P}}(t_{2P}, q_{execute}, q_{send}, q_{reveal}) + \frac{p(\kappa)^2}{2^{k+1}}.$$
Games $G_2^1, \ldots, G_n^2$ are defined similarly. The similar reasoning of existence of a distinguisher between games $G^i_1$ and $G^{i+1}_1$ works. When we sum all inequalities on the left side and on the right side,

$$|\Pr[\text{Succ}_2^n] - \Pr[\text{Succ}_0^n]| \leq n \cdot p(\kappa) \mathsf{Adv}_{AKE_{2P}}(t_{2P}, q_{execute}, q_{send}, q_{reveal}) + \frac{np(\kappa)^2}{2^{i_k+1}}.$$

In this part we simulate gradual replacement of values $K_i$ by random values in the games $G^i_2, i = 1 \ldots n$. 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 $G^2_2$:** We simulate something 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

$$|\Pr[\text{Succ}_2^1] - \Pr[\text{Succ}_0^n]| \leq p(\kappa) \mathsf{Adv}_{DDH_{G,A_{DDH}}}(t_{DDH}).$$

To prove this inequality, suppose that there exist a distinguisher $A_D$ which can distinguish these two games. We can use this distinguisher to construct an adversary $A_{DDH}$, which can solve DDH problem, with use of similar hybrid argument as in previous games: we define a distribution $H^i_2$, $i \in \{0, 1, \ldots p(\kappa)\}$. In the distribution $H^i_2$ the values $K_1$ for instances $\Pi^i_1, j \leq i$ are chosen randomly and the values $K_1$ for instances $\Pi^i_1, j > i$ are computed as in the previous game. We assume that distinguisher $A_D$ constructs $p(\kappa)$ sessions for some polynomial $p(\cdot)$ during simulation.

**Adversary** $A_{DDH}(u, v, w)$

1. $A_{DDH}$ chooses a random bit $b$ and an index $j$.

2. $A_{DDH}$ answers oracle queries of the distinguisher $A_D$ 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 $\Pi^i_1$, 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 $A_{DDH}$ knows values $sk_1, sk_2$ – he has chosen them randomly. Then he simulates that $P_1$ sends a message $(ID(P_1)||E_{sk_1}(u))$ and $P_2$ sends a message $(ID(P_2)||E_{sk_2}(v))$ then $K_1$ is set to $w$, $K_2 = v^{x_2}$, $K_n = u^{x_n}$. 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 $\Pi^i_1$, where $l < j$ then $A_{DDH}$ 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 $sk_1^i$, $i = 1 \ldots n$ 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 $P_l$ has form $\Pi^l_1$, where $l > j$ then $A_{DDH}$ 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. $A_{DDH}$ returns a $b \leftarrow A_D$

If $(u, v, w)$ from adversary’s input is a DDH triple and the index $j = 0$, the view of the distinguisher $A_D$ is the same as in the game $G^n_1 (H^n_2)$. If $(u, v, w)$ is not a DDH triple and the index $j = p(k)$, the view of $A_D$ is the same as in the game $G^n_2 (H^n_2)$. Thus the advantage of $A_{DDH}$ is at least as great as $\frac{1}{p(k)}$ of the advantage of $A_D$ (we skip the detailed reasoning).

$$| \Pr[Succ^n_2] - \Pr[Succ^n_1]| = p(k) \text{Adv}^{DDH}_{G^n, A_{DDH}}(t_{DDH})$$

The games $G^n_2, \ldots, G^n_m$ are defined similarly. When we sum inequalities, we have

$$| \Pr[Succ^n_2] - \Pr[Succ^n_1]| = n \cdot p(k) \cdot \text{Adv}^{DDH}_{G^n, A_{DDH}}(t_{DDH}).$$

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

$$\Pr[Succ_3] = \Pr[Succ^n_2].$$

This claim follows from the view of an adversary in this two games. In the game $G^n_2$ 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 $\text{Chall}^\beta(\cdot)$ returns vectors $(\zeta_1, \ldots, \zeta_n, \gamma_1, \ldots, \gamma_n)$ from the distribution $PDH^n_{\Lambda_2}$. New vectors are returned in every session, however the same vectors are returned in queries on the same session. For randomly chosen $(\alpha_1, \ldots, \alpha_n)$, $\alpha_i \leftarrow Z^*_q$, vectors $(\zeta^\alpha_1, \ldots, \zeta^\alpha_n, \gamma^\alpha_1, \ldots, \gamma^\alpha_n)$ have equal distribution to the original $(\zeta_1, \ldots, \zeta_n, \gamma_1, \ldots, \gamma_n)$. We use this property for application of random self-reducibility of the PDDH problem. The decryption is changed as follows:

- If instance of $P_i$ has form $\Pi^l_1$, where $l > j$ then $A_{DDH}$ 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.

- $D_{sk_i}(z^*)$ — if $(sid^l_i, i, sk_i) \in \Lambda_{2P}$, $(\zeta_1, \ldots, \zeta_n, \gamma_1, \ldots, \gamma_n) \leftarrow \text{Chall}^\beta(sk_1, \ldots, sk_n)$ (the arguments of $\text{Chall}^\beta$ can be found in the $\Lambda_{2P}$ list sharing the same value of the session ID), we choose $\alpha_i \leftarrow Z^*_q$ randomly and compute $z_i = \zeta_i^{\alpha_i}$. If $(\cdot, \cdot, \cdot, sk_i, z_i, \cdot) \in \Lambda_{\mathcal{E}}$, then we stop the simulation, adversary wins. Otherwise we add record $(dec, sid^l_i, i, \alpha_i, sk_i, z_i, z^*)$ 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 $\Lambda_E$ 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 $Send_1^j$, $Send_2^j$ and $Send_3$ queries. First, encryption of messages in the second step of the protocol is changed during simulation of $Send_1^j$. The instance $\Pi_i^j$ chooses $z_i^* \leftarrow G$ randomly and computes $z_i = D_{sk_i}(z_i^*)$ as in the previous game. Then $\Pi_i^j$ sends a message $\langle ID(P_i) || z_i^* \rangle$. Therefore $Send_1^j$ queries in the second step of the MLHL lead to adding of $\alpha_i$ to the list $\Lambda_E$. Simulation ends if

- $(enc, \perp, \perp, \perp, sk_i, \cdot, z_i^*) \in \Lambda_E$, 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 $(enc, \perp, \perp, \perp, sk_i, \cdot, z_i^*) \in \Lambda_E$). The probability of this event is $q_E/2^{ik}$. In this case we stop the simulation, the adversary wins.

- $(dec, \perp, \perp, \perp, sk_i, z_i, z_i^*) \in \Lambda_E$. 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 $\langle ID(P_i) || z_i^* \rangle$ during simulation of $Send_2^j$, 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 below.

- $z_i^*$ is response of the encryption oracle $E_{sk_i}$, while $sk_i$ is a 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 $q_E/2^{ik}$.

- $z_i^*$ 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 $z_i' = D_{sk_i}(z_i^*)$, then we compute $z_i^{**} = E_{sk_{i+1}}(z_i')$ and send to the user $P_{i+1}$ a message $\langle ID(S) || ID(P_i) || z_i^{**} \rangle$. 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 $Adv_{\text{MAC}, \text{forge}}(t')$, which is negligible.

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

- if \((\cdot, \cdot, \cdot, s_{k_{i+1}}, z_i, \cdot) \notin \Lambda_\epsilon\) and \((dec, sid^j_i, i, \alpha_i, sk_i, z_i, \cdot) \in \Lambda_\epsilon\) (this record was added in the simulation described above by the instance \(\Pi^j_i\)), then we choose \(z^{**} \leftarrow G, i\) if \((\cdot, \cdot, \cdot, s_{k_{i+1}}, \cdot, z^{**}) \notin \Lambda_\epsilon\), we return \(z^{**}\) and add record \((enc, sid^j_i, i, \bot, s_{k_{i+1}}, z, z^{**})\) into \(\Lambda_\epsilon\), else we stop the simulation and the adversary wins.

- if \((enc, \bot, \bot, \bot, s_{k_{i+1}}, z_i, \cdot) \in \Lambda_\epsilon\), we stop the simulation and the adversary wins. This case occurs if the adversary asked for encryption of value \(z_i\) with key \(s_{k_{i+1}}\). The probability of this event is \(q_\epsilon/2^{|k|}\).

- if \((enc, sid^j_i, i + 2, \bot, s_{k_{i+1}}, z_i, z^*) \in \Lambda_\epsilon\), 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 \(\Pi^j_{i+1}\) (sent with the instance \(\Pi^j_{i+2}\)) happens, while the value was encrypted with the key \(s_{k_{i+1}}\).

- if \((dec, sid^j_i, i + 1, \alpha_{i+1}, s_{k_{i+1}}, z_i, z^*) \in \Lambda_\epsilon\), we return \(z^*\). This case can occur by simulation of Send queries of the instance \(\Pi^j_{i+1}\) in the second round.

\[ E_{sk_{i-1}}(z_i) \] — similar to the previous case.

The simulation of Send_{3}, when \(\Pi^j_i\) 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 \(z_{i-1} = D_{sk_i}(z_{i-1}^*)\) and \(z_{i+1} = D_{sk_i}(z_{i+1}^*)\). 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 below.

- one or both \(z_{i-1}^*\) and \(z_{i+1}^*\) is/are the answer from query on the encryption oracle \(E_{sk_i}\), while \(sk_i\) is correct key of the instance \(\Pi^j_i\) in the session \(j\) (thus adversary guessed a password and used its for encryption of data from server). This event occurs with probability \(q_\epsilon/2^{|k|}\). 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 \(\text{Adv}^{\text{MAC-forge}}_{M,\Lambda_M}(t_M)\), which is negligible.

If messages were sent as we simulate them, we have \(z_i = \zeta^\alpha_i, z_{i-1} = \zeta^\alpha_{i-1}, z_{i+1} = \zeta^\alpha_{i+1}\) and we can compute

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

\[ w_i = K_{i-1} \oplus K_i, \tau_i = \text{Mac}_{K_i}(w_i) \]

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.

\[ |\Pr[Succ_5] - \Pr[Succ_4]| \leq 4qE/2^{l_k} + 2\text{Adv}_{\text{MAC-forgery}}^{\text{MAC}}(t_M) \]

**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 $PDH^2_{\text{pk}}$. Clearly holds that

\[ |\Pr[Succ_6] - \Pr[Succ_5]| \leq p(\kappa) \cdot \text{Adv}_{G,A_{\text{DDH}}}^{\text{DDH}}(t_{DDH}), \]

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 $\Pr[Succ_7] = 1/2$, because the session key is randomly chosen and independent from the previous messages. When we sum all (in)equalities of games, we have:

\[ |\Pr[Succ_7] - \Pr[Succ_0]| \leq \frac{(qE + qD)^2}{2|G|} + \frac{p(\kappa) \cdot n \cdot qD}{2^l_k} + 2\text{Adv}_{\text{MAC-forgery}}^{\text{MAC}}(t_M) \]

\[ + p(\kappa) \cdot n \cdot \text{Adv}_{2P,A_{2P}}^{\text{AKE}}(t_{2P}, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}}) \]

\[ + \frac{np(\kappa)^2}{2^{l_k+1}} + p(\kappa) \cdot n \cdot \text{Adv}_{G,A_{\text{DDH}}}^{\text{DDH}}(t_{DDH}) + 4qE/2^{l-k} \]

\[ + p(\kappa) \cdot \text{Adv}_{G,A_{\text{PDH}}}^{\text{PDH}}(t_{DDH}) \]

\[ \text{Adv}_{\text{MLHL,A_{MLHL}}}^{\text{AKE}}(A) \leq 2\left( \frac{(qE + qD)^2}{2|G|} + \frac{p(\kappa) \cdot n \cdot qD}{2^l_k} + 2\text{Adv}_{\text{MAC-forgery}}^{\text{MAC}}(t_M) \right) \]

\[ + p(\kappa) \cdot n \cdot \text{Adv}_{2P,A_{2P}}^{\text{AKE}}(t_{2P}, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}}) \]

\[ + \frac{np(\kappa)^2}{2^{l_k+1}} + 2p(\kappa) \cdot n \cdot \text{Adv}_{G,A_{\text{DDH}}}^{\text{DDH}}(t_{DDH}) + 4qE/2^{l-k} \right) . \]