Security of A Server-Assisted Group Password-Authenticated Key Exchange Protocol

Shaoquan Jiang and Guang Gong

Department of Electrical and Computer Engineering
University of Waterloo, Waterloo, ON, N2L 3G1, CANADA
Email: {jiang,ggong}@calliope.uwaterloo.ca

Abstract. A server-assisted n-party password-authenticated key exchange (nPAKE$^+$) protocol is to establish a session key among n clients, each of which shares an *independent* password with a common server. However, to our knowledge, a provably secure nPAKE$^+$ protocol for a general or dynamic n has not been previously obtained. Recently, an efficient but *unproved* protocol for a general and dynamic n was reported by Wan [20]. In this paper, we show that their protocol is secure in the random oracle and ideal cipher model.

1 Introduction

In secure communications, key exchange (KE) probably is one of the most important issues, where through an interactive procedure, the involved parties establish a temporary session key, and then they use this key to protect the subsequent communications. In the literature, there are two types of KE protocols. In the first type, each party holds a high entropic secret (e.g., a signing key of a digital signature). Research along this line has been well studied, see [4, 9, 10] for examples. In the second type, the involved parties only hold a human memorizable password. The password dictionary could be small. So, an exhaustive search attack (or a *dictionary attack*) is widely believed to be doable. Thus, it is not appropriate to directly apply the first type of protocol to the second setting without a rigorous security proof. In this work, we will investigate the second type of key exchange protocol in the n-party setting. In this case, the n-party shares a single password, or each party shares an independent password with a trusted common server, or each pair of the parties share an independent password. The third setting is not desirable as it requires each party to store many passwords. Denote the protocol in the first setting by an nPAKE^0 protocol and the protocol in the second setting by an nPAKE$^+$ protocol. Although nPAKE^0 protocols have many applications, nPAKE$^+$ protocols seem to have their own merits. For example, compromising one party's password will trivially break an nPAKE^0 protocol while it only has a local effect on an nPAKE$^+$ protocol. An nPAKE$^+$ protocol allows a subgroup to execute the protocol while this is impossible in an nPAKE^0 protocol. Thus, it is quite interesting to study a secure nPAKE$^+$ protocol.

1.1 Related Work

Two-party password-authenticated key exchange (2PAKE) protocols were first studied by Bellovin and Merritt [7]. Since then, it has been an active research topic (see [6, 14, 16, 11] for examples).

In the literature, many nPAKE^0 protocols have been reported (3PAKE^0 [12, 13, 17] and nPAKE^0 [2]). Unfortunately, none of them have been provably secure. To our knowledge, the only known provably secure nPAKE^0 protocol is due to Bresson *et al.* [8]. The research status of nPAKE$^+$ protocols is more discouraging. The only known provably secure protocol is a 2PAKE$^+$ protocol by

Abdalla *et al.* [1]. Their protocol is obtained by appropriately composing two 2PAKE^0 protocols. Thus, it is not efficient. Recently, an efficient but *unproven* nPAKE$^+$ protocol for a general and dynamic n was proposed by Wan [20]. Their construction is based on a Diffie-Helleman key tree, motivated by a *one-way function tree framework* from [18].

1.2 Contribution

In this paper, we first formalize a security model for the nPAKE$^+$ problem. Our model is an extension of the two-party model [6]. Then, we prove that the nPAKE$^+$ protocol [20] is secure for a general and dynamic n (i.e., n can be different for distinct executions). Our proof is in the random oracle and ideal cipher model.

This paper is organized as follows. We present the security model in Section 2. In Section 3, we prove properties for a Diffie-Hellman tree. We review the protocol [20] in Section 4 and prove the security in Section 5.

2 Security Model

In this section, we introduce a security model for server-assisted password-authenticated n-party key exchange (nPAKE$^+$) protocols. This model is extended from the two-party model in [6]. We start with the following notation.

- D: a password dictionary of a *polynomial* size with uniform distribution.
- P: some party, either a client C or a server S. If it is a server, then it could individually share a password with a set of clients. For simplicity, we assume that a client only belongs to *one* server. When we use P_i to denote a party, we do *not* mean i is a *global* label (instead, it is for our reference use only).
- P^l : protocol instance l within party P. We require that l be unique within P in order to distinguish local instances.
- $\langle \mathsf{P}_1^{l_1}, \cdots, \mathsf{P}_n^{l_n} \rangle$: An *ordered* list of instances which are supposed to jointly run an n-party protocol execution in a manner that $\mathsf{P}_j^{l_j}$ plays the role of Player j.
- $\mathbf{ed}_i^{l_i}$: *essential data* for instance $\mathsf{P}_i^{l_i}$, where we assume P_i is clear from the context. This is a notion similar to *session identifier* [6]. But we restrict $\mathbf{ed}_i^{l_i}$ to be a portion of messages received by $\mathsf{P}_i^{l_i}$ plus (maybe) some local random bits.
- $Flow_d$: If m is the d-th message exchanged between the sender and the receiver of m, then we say m is a $Flow_d$ message.
- $\mathbf{pid}_i^{l_i}$: an *ordered* list of parties, say $\langle P_1, \cdots, P_n \rangle$, indicated in the *view* of $\mathsf{P}_i^{l_i}$ (a view consists of local random bits and the messages received so far) such that P_t plays the role of Player t.

Valid View. Consider an instance list $Ins = \langle \mathsf{P}_1^{l_1}, \cdots, \mathsf{P}_n^{l_n} \rangle$ and an instance $X = \mathsf{P}_t^{l_t}$ for some $1 \le t \le n$. We say X has a *valid view* on $m \in \mathbf{ed}_t^{l_t}$ w.r.t. Ins if m was computed and sent from the instance in Ins (called the *originator* of m) as specified in the protocol. To be unambiguous, the originator for each $m \in \mathbf{ed}_t^{l_t}$ should be explicitly defined in the security proof. By default, we assume local random bits *originate* from the underlying instance itself.

Partnering. Two instances X and Y are said to be *partnered* if there exists an instance list $Ins = \langle \mathsf{P}_1^{l_1}, \cdots, \mathsf{P}_n^{l_n} \rangle$ with $X = \mathsf{P}_i^{l_i}$ and $Y = \mathsf{P}_j^{l_j}$ for some i, j such that (1) $\mathbf{pid}_i^{l_i} = \mathbf{pid}_j^{l_j}$; (2) $\mathsf{P}_i^{l_i}$

(resp. $\mathsf{P}_j^{l_j}$) has a valid view on $\mathbf{ed}_i^{l_i}$ (resp. $\mathbf{ed}_j^{l_j}$) with respect to *Ins*.

Adversarial Model. *Before the protocol starts*, the adversary can specify a set of clients for corruption. As a result, he will receive the passwords of these clients and take the actions of these parties in the future. This models the setting: some clients collude together to compromise the system. We stress that we fully trust a server as compromising it will trivially break the system.

After the protocol starts, the adversary is allowed to fully control the external network. He can inject, modify, block and delete messages at will. He can also request any session key adaptively. Formally, he can adaptively query the following oracles.

- **Execute**$(\mathsf{P}_1^{l_1}, \cdots, \mathsf{P}_n^{l_n})$: If some instance $\mathsf{P}_t^{l_t}$ has already existed, the output is $\perp$; otherwise, a protocol execution among $Ins = \langle \mathsf{P}_1^{l_1}, \cdots, \mathsf{P}_n^{l_n} \rangle$ takes place such that the oracle plays the roles of uncorrupted parties and the adversary plays the roles of corrupted parties (if any). The only constraint on the adversary is that he must behave *consistently* in view of oracle (e.g., messages from honest party to honest party should not be modified). This oracle call models an eavesdropping attack (possibly from colluding clients).
- **Send**(d, i, l_i, j, M) : Assume $\mathbf{pid}_i^{l_i} = \langle P_1, \cdots, P_n \rangle$. P_i should not be corrupted as a corrupted party is fully controlled by the adversary. When this oracle is called, message M is sent to instance $\mathsf{P}_i^{l_i}$ as $Flow_d$, claiming it comes from party P_j. The oracle output is whatever $\mathsf{P}_i^{l_i}$ returns. The oracle reflects a threat from the man-in-the-middle attack.
- **Reveal**(i, l_i) : Again assume P_i is clear from the context and uncorrupted. When this oracle is called, it outputs the session key of instance $\mathsf{P}_i^{l_i}$ if a session key has been defined for $\mathsf{P}_i^{l_i}$; otherwise, it outputs $\perp$. This oracle reflects the threat from a session key loss attack.
- **Test**(i, l_i) : Again assume P_i is clear in the context and uncorrupted. The adversary is allowed to query it only once. The queried instance must have defined the session key, and no party in $\mathbf{pid}_i^{l_i}$ has been corrupted. Furthermore, this instance as well as any partnered instance (if it exists) will never be issued a **Reveal** query. When this oracle is called, it flips a fair coin b. If $b = 1$, then the session key of $\mathsf{P}_i^{l_i}$ is provided to the adversary; otherwise, a random number of the same length is provided.

Having defined adversary behaviors, we come to characterize the terms for security.

Completeness.

- ◇ For any uncorrupted P_i, $\mathbf{ed}_i^{l_i}$ is unique within P_i, and the session key $sk_i^{l_i}$ is determined by $\mathbf{ed}_i^{l_i}$.
- ◇ Assume two partnered instances X and Y complete and accept such that (1) each $\mathsf{P}_t^{l_t}$ in the underlying Ins (see the definition of partnering) has a valid view on $m \in \mathbf{ed}_t^{l_t}$ w.r.t Ins as long as in the protocol specification m is defined before X or Y completes[1], (2) no party in Ins has been corrupted. Then X and Y compute the same session key.

Authentication. Let $\mathsf{P}_i^{l_i}$ be any *client* instance with $\mathbf{pid}_i^{l_i} = \langle P_1, \cdots, P_n \rangle$ such that no P_t has been corrupted. When $\mathsf{P}_i^{l_i}$ completes and accepts, there exists a unique list $Ins = \langle \mathsf{P}_1^{l_1}, \cdots, \mathsf{P}_n^{l_n} \rangle$ (containing $\mathsf{P}_i^{l_i}$) such that for each j, $\mathsf{P}_j^{l_j}$ has a *valid view* on $m \in \mathbf{ed}_j^{l_j}$ w.r.t. Ins as long as in the protocol specification m is defined before $\mathsf{P}_i^{l_i}$ completes. We say the authentication property is broken if this condition is violated.

[1] Here we assume a protocol specification is described by following the execution procedure.

Privacy. At the end of the attack, the adversary outputs a guess bit b' for the bit b chosen in the **Test** session. He is said to succeed if $b' = b$.

Remark 1. Note in the authentication property, if no client instance accepts, we do not promise any authentication w.r.t. Ins (in view of a server). This is reasonable: (1) in this case no session key needs to be protected; (2) adding the authentication (in view of a server) could render the protocol more vulnerable to attacks. Indeed, in the protocol at Section 4, the server never stores an internal state. If we maintain such an authentication by including one return message to server, then the server has to store an internal state in order to make a decision later. This renders the resulting protocol more complex and more vulnerable to denial of service attack.

We use random variable $\mathbf{Succ}(\mathcal{A}, \Gamma)$ to denote the success event for an adversary $\mathcal{A}$ in a key exchange protocol Γ. The advantage of $\mathcal{A}$ is defined to $\mathbf{Adv}(\mathcal{A}) := 2\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma)] - 1$.

Now we are ready to provide a formal definition of security.

Definition 1. *A password-based server-assisted group key exchange protocol Γ is said to be secure if it satisfies*

- *Completeness.*
- *Authentication and Privacy. If adversary $\mathcal{A}$ makes at most Q_{send} queries to* **Send** *oracle, then*

$$\mathbf{Adv}(\mathcal{A}) < \frac{n_0 Q_{send}}{|D|} + \mathbf{negl}(\kappa), \tag{1}$$

where D is the password dictionary, n_0 is an upperbound on the number of clients involved in one protocol execution, and κ is the security parameter.

3 Foundation: A Diffie-Hellman Key Tree

In this section, we introduce the Diffie-Hellman key tree, a building block of the protocol in [20].

Let p, q be large primes with $p = 2q + 1$; G_q be the subgroup of order q in $\mathbb{Z}_p^*$; $g \in G_q \backslash \{1\}$. Let $\mathbf{T}$ be an arbitrary binary tree such that each internal node has exactly two children. We call this type of tree a *regular* binary tree. Denote the leaves of $\mathbf{T}$ from the left-most to the right-most by $v_1, \cdots, v_n$. An internal node is labelled as $v_{a,b}$ if v_a (resp. v_b) is the left-most (resp. the right-most) leaf that is rooted at this node. In the future, when we use v_s to represent a node in $\mathbf{T}$, we mean $s \in \{1, \cdots, n\}$ (in case of a leaf node) or $s = (a, b)$ (in case of an internal node $v_{a,b}$). For each node $v_s \in \mathbf{T}$, we associate it with two keys: exponential key (or briefly Exp-K) K_s and public key (or Pub-K) K'_s, where $K'_s = g^{|K_s|_p}$ and $|\cdot|_p : G_q \to \mathbb{Z}_q$ is a function[2]:

$$|x|_p = \begin{cases} x & \text{if } 1 \le x < q, \\ p - x & \text{if } q + 2 \le x < p, \\ 0 & \text{otherwise.} \end{cases}$$

[2] Note that G_q is exactly the set of quadratic residues in $\mathbb{Z}_p^*$. Note $\left(\frac{q(q+1)}{p}\right) = \left(\frac{1+q^{-1}}{p}\right) = \left(\frac{-1}{p}\right) = -1$ as $p = 3 \pmod 4$. It follows that either $q \in G_q$ or $q+1 \in G_q$ but not both. More precisely, by *Law of Quadratic Reciprocity* [15], we have that $q \in G_q$ if $q = 1 \pmod 4$ and $q + 1 \in G_q$ if $q = 3 \pmod 4$. Further notice that $\left(\frac{p-x}{p}\right) = -1$, $\forall x \in G_q$. Therefore, we have that $|\cdot|_p$ is a 1-1 and onto mapping from G_q to $\mathbb{Z}_q$.

Furthermore, the Exp-K K_s for a node $v_s \in \mathbf{T}$ is recursively defined as follows. If α, β be the Exp-Ks on its left child and right child, then $K_s := g^{|\alpha|_p |\beta|_p}$. So, in Figure 1, $K_{1,8} = g^{|K_{1,4}|_p |K_{5,8}|_p}$.

In addition, for each $v_{i,i+1} \notin \mathbf{T}$ $(1 \leq i \leq n-1)$, define $K_{i,i+1} = K_i'^{|K_{i+1}|_p} = g^{|K_i|_p |K_{i+1}|_p}$. For example, in Figure 1, such keys are $K_{2,3}, K_{4,5}, K_{6,7}$.

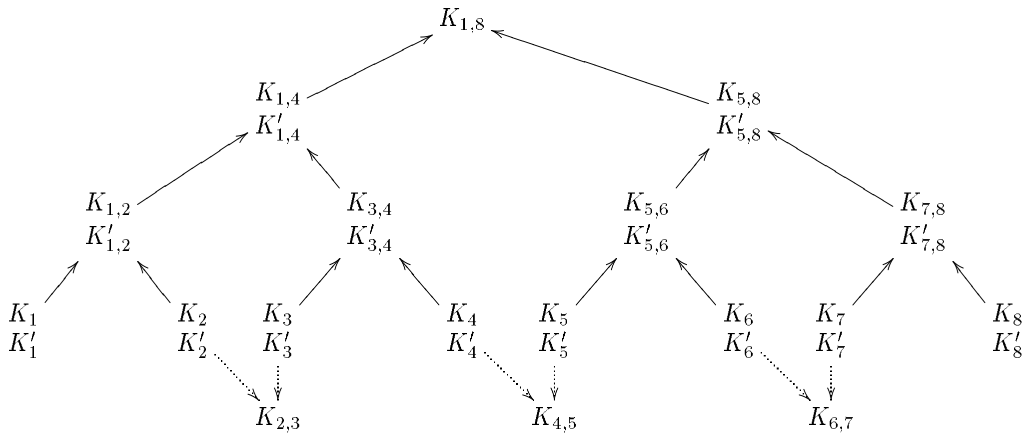

Fig. 1. Example for a Diffie-Hellman Key Tree ($v_{2,3}, v_{4,5}, v_{6,7}$ are not nodes of $\mathbf{T}$)

Define

$$\begin{aligned} \mathrm{DH}(\mathbf{T}) = &\{K'_s : s \neq (1,n), v_s \in \mathbf{T}\} \cup \{K_{1,n}\} \cup \{K_{i,i+1} : v_{i,i+1} \notin \mathbf{T}, 1 \leq i < n\}, \\ &\text{where the leaf keys } K_1, K_2, \cdots, K_n \text{ are uniformly at random in } G_q; \\ \mathrm{R}(\mathbf{T}) = &\{R_s : s \neq (1,n), v_s \in \mathbf{T}, R_s \leftarrow G_q\} \cup \\ &\{R_{1,n} : R_{1,n} \leftarrow G_q\} \cup \{R_{i,i+1} : v_{i,i+1} \notin \mathbf{T}, 1 \leq i < n, R_{i,i+1} \leftarrow G_q\}. \end{aligned}$$

In Figure 1,

$$\mathrm{DH}(\mathbf{T}) = \{K'_1, \cdots, K'_8\} \cup \{K'_{1,2}, K'_{3,4}, K'_{5,6}, K'_{7.8}, K'_{1,4}, K'_{5,8}\} \cup \{K_{1,8}\} \cup \{K_{2,3}, K_{4,5}, K_{6,7}\}.$$

The following lemma is important to the security of the protocol.

Lemma 1. *Let* $\mathbf{T}$ *be any regular binary tree. Then under the* DDH *assumption,* $\mathrm{DH}(\mathbf{T})$ *is indistinguishable from* $\mathrm{R}(\mathbf{T})$.

Proof. We use induction on n. If $n = 2$, $\mathrm{DH}(\mathbf{T})$=$\{K'_1, K'_2, K_{1,2}\}$ and $\mathrm{RH}(\mathbf{T})$= $\{R_1, R_2, R_{1,2}\}$. The conclusion directly follows from the DDH assumption, as $|\cdot|_p : G_q \to \mathbb{Z}_q$ is a 1-1 and onto mapping. Assume that the conclusion holds for $n \leq l$. We consider a regular binary tree $\mathbf{T}$ of $l+1$ leaves. Without loss of generality, assume v_l and v_{l+1} are siblings (if v_i, v_{i+1} are siblings for other i, the proof is similar). If we remove nodes v_l and v_{l+1}, we get another regular tree $\mathbf{T}'$. Let $\mathrm{DR}(\mathbf{T})$ be defined as $\mathrm{DH}(\mathbf{T})$ except that $K_{l,l+1}, K_{l,l-1}$ are taken as uniformly at random in G_q (instead of $g^{|K_l|_p |K_{l+1}|_p}$ and $g^{|K_l|_p |K_{l-1}|_p}$, respectively). To prove the lemma, it suffices to prove that the indistinguishability holds for $\mathrm{DH}(\mathbf{T})$ and $\mathrm{DR}(\mathbf{T})$, $\mathrm{DR}(\mathbf{T})$ and $\mathrm{R}(\mathbf{T})$, respectively.

If $\mathrm{DR}(\mathbf{T})$ and $\mathrm{DH}(\mathbf{T})$ are distinguishable by $\mathcal{A}$, then a DDH distinguisher $\mathcal{A}'$ is built as follows. Upon input (α, β, γ), $\mathcal{A}'$ takes $(K'_l, K'_{l+1}, K_{l,l+1}) \leftarrow Rud_1(\alpha, \beta, \gamma)$ and $(K'_l, K'_{l-1}, K_{l,l-1}) \leftarrow$

$Rud_0(K'_l, K'_{l+1}, K_{l,l+1})$ (see Appendix 1 for more details). Then he follows the definition of Diffie-Hellman Key tree to compute $\{K'_s : v_s \in \mathbf{T}, s \neq (1, l+1)\}\} \cup \{K_{1,l+1}\} \cup \{K_{i,i+1} : v_{i,i+1} \notin \mathbf{T}, 1 \leq i < n\}$, except $K'_l, K'_{l+1}, K_{l,l+1}, K_{l,l-1}$ are defined as above. Finally, he provides this tuple to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ does. When (α, β, γ) is a DH tuple, the input to $\mathcal{A}$ is DH($\mathbf{T}$); otherwise, it is DR($\mathbf{T}$). Thus, $\mathbf{Adv}(\mathcal{A}') = \mathbf{Adv}(\mathcal{A})$, non-negligible, contradicting the DDH assumption.

If DR($\mathbf{T}$) is distinguishable from R($\mathbf{T}$) by $\mathcal{A}$, a distinguisher $\mathcal{D}$ for DR($\mathbf{T}'$) and R($\mathbf{T}'$) can be built as follows. Upon input MH($\mathbf{T}'$) (either DR($\mathbf{T}'$) or R($\mathbf{T}'$)), $\mathcal{D}$ removes the entry corresponding to $v_{l-1,l}$ from MH($\mathbf{T}'$) and then regards the remaining entries as partial entries of DH($\mathbf{T}$) by embedding $\mathbf{T}'$ back to $\mathbf{T}$. This leaves $K'_l, K'_{l+1}, K_{l-1,l}$ undefined. He then simply defines $K'_l, K'_{l+1}, K_{l-1,l}$ to be uniformly at random. $\mathcal{D}$ provides the obtained tuple to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ does. If MH($\mathbf{T}'$) is DH($\mathbf{T}'$), then the input to $\mathcal{A}$ is DR($\mathbf{T}$); otherwise, it is R($\mathbf{T}$). Thus, $\mathcal{A}$ and $\mathcal{D}$ have identical (non-negligible) advantages, contradicting to the induction assumption. $\square$

The following notations are important in the protocol. For $i = 1, \cdots, n$, define

$$R_i = \{K'_s : v_s \text{ hangs off the path from } v_i \text{ to } v_{1,n} \text{ on the right side, and } v_s \text{ is not a leaf}\},$$
$$L_i = \{K'_s : v_s \text{ hangs off the path from } v_i \text{ to } v_{1,n} \text{ on the left side, and } v_s \text{ is not a leaf}\},$$

where v_s is said to **hang off** the path from v_i to $v_{1,n}$ on the right (resp. left) side, if v_s is not on this path but its parent $v_{s'}$ is on the path such that v_s is the right (resp. left) child of $v_{s'}$. For example, in Figure 1, we have $R_1 = R_2 = \{K'_{3,4}, K'_{5,8}\}$, $R_3 = R_4 = \{K'_{5,8}\}, R_5 = R_6 = \{K'_{7,8}\}, R_7 = R_8 = L_1 = L_2 = nil$, $L_3 = L_4 = \{K'_{1,2}\}, L_5 = L_6 = \{K'_{1,4}\}, L_7 = L_8 = \{K'_{5,6}, K'_{1,4}\}$.

The following simple lemma constitutes the design basis of the protocol.

Lemma 2. *(i) R_{i-1} (resp. L_{i+1}) is computed by only using R_i (resp. L_i) together with K_i and $\{K'_j : 1 \leq j \leq n\}$. (ii) Either $R_i = R_{i-1}$ or $L_i = L_{i+1}$. (iii) If d_i is the distance from v_i to the root $v_{1,n}$, then $K_{1,n}, L_{i+1}, R_{i-1}$ can be computed from L_i, K_i, R_i together with $\{K'_j : 1 \leq j \leq n\}$ using $d_i + 1$ exponentiations only.*

Proof. (i) Let u be the nearest common ancestor of v_i and v_{i-1}. Then v_i is rooted at the right child v_r of u and v_{i-1} is rooted at the left child v_l of u. Furthermore, since v_i and v_{i-1} are consecutive leaf nodes, any node v_s hanging off the path from v_i to v_r must be on the right side. So K'_s is either in R_i or $\{K'_1, \cdots, K'_n\}$. Therefore, with K_i, R_i and $\{K'_1, \cdots, K'_n\}$, one can recursively compute K_e for each v_e on the path from v_i to u_r. Specifically, if $v_{e'}$ and $v_{e''}$ are the left and right children of v_e, then $K_e = (K'_{e''})^{|K_{e'}|_p}$. Let u have a distance rd_i from v_i. It follows that it requires at most rd_i modular exponentiations to compute K'_r and K_r from K_i, R_i and $\{K'_1, \cdots, K_n\}$. On the other hand, no node hanging off the path from v_{i-1} to v_l is on the right side, as v_i, v_{i-1} are consecutive leaf nodes. Thus, $R_{i-1} = \{K'_s : v_s \text{ hangs off the path from } v_l \text{ to } v_{1,n} \text{ on the right side, and } v_s \text{ is not a leaf }\}$, while the latter is a subset of $R_i \cup \{K'_r\} \cup \{K'_1, \cdots, K'_n\}$. Therefore, R_{i-1} can be obtained from K_i, R_i and $\{K'_1, \cdots, K'_n\}$ using rd_i modular exponentiations. Similarly, we can prove L_{i+1} can be computed from $L_i, K_i, \{K'_1, \cdots, K'_n\}$.

(ii) In the proof of (i), if $v_r = v_i$, we have $R_{i-1} = R_i$; otherwise, v_{i+1} must be rooted at v_r. In this case, v_i is the only node hanging off the path from v_{i+1} to v_r on the left side. Thus, $L_i = L_{i+1}$.

(iii) Assume $L_i = L_{i+1}$. After computing K_r, one can recursively compute K_e for each v_e on the path from v_r to $v_{1,n}$, since Pub-K of the right child of v_e is in $R_i \cup \{K'_1, \cdots, K'_n\} \cup \{K'_r\}$. This needs $d_i - (rd_i - 1)$ modular exponentiations. Thus, it requires $d_i + 1$ modular exponentiations to compute L_{i+1}, R_{i-1} and $K_{1,n}$. Similarly, if $R_i = R_{i-1}$, the conclusion holds too. $\square$

Consider $i = 5$ in Figure 1. $R_4 = \{K'_{5,8}\}$ can be computed from $R_5 (= \{K'_{7,8}\})$, $\{K_5\}$ and $\{K'_1, \cdots, K'_8\}$ as follows. $K_{5,6} = (K'_6)^{|K_5|_p}, K_{5,8} = (K'_{7,8})^{|K_{5,6}|_p}, K'_{5,8} = g^{|K_{5,8}|_p}$. $L_6 = L_5$ holds. Furthermore, $K_{1,8} = (K'_{1,4})^{|K_{5,8}|_p}$, where $K'_{1,4} \in L_5$. Thus, 4 $(= d_5{+}1)$ modular exponentiations are used to compute $L_6, R_4, K_{1,8}$ from L_5, R_5, $\{K_5\}$, and $\{K'_1, \cdots, K'_n\}$, among which 3 $(= rd_5 + 1)$ modular exponentiations are used to compute R_4.

4 Review of the Protocol [20]

In this section, we will review the nPAKE$^+$ protocol [20]. Assume clients $C_1, \cdots, C_n$ want to establish a common session key with the help of a server S. Each C_i shares an independent password p_i with S. We assume Diffie-Hellman tree structure $\mathbf{T}$ with n leaves is fixed with depth at most $\lceil \log n \rceil$. Assume p and q are large primes with $p = 2q + 1$ and G_q is the subgroup of order q in $\mathbb{Z}_p^*$. Let $g \in G_q \backslash \{1\}$. Thus, $\langle g \rangle = G_q$. Assume $F_k()$ is a message authentication code with a secret key k, $f : G_q \to \{0,1\}^c$ is a hash function, and $(\cdot)_{p_i} : G_q \to G_q$ is an encryption algorithm E with secret key p_i (for simplicity, we assume p_i is in the key space of E). The protocol is as follows, see Figure 2 for the message transmission in the case $n = 5$. Note *the basic implementation check of finite field membership in the protocol is important. For simplicity, we always assume it is successfully verified.*

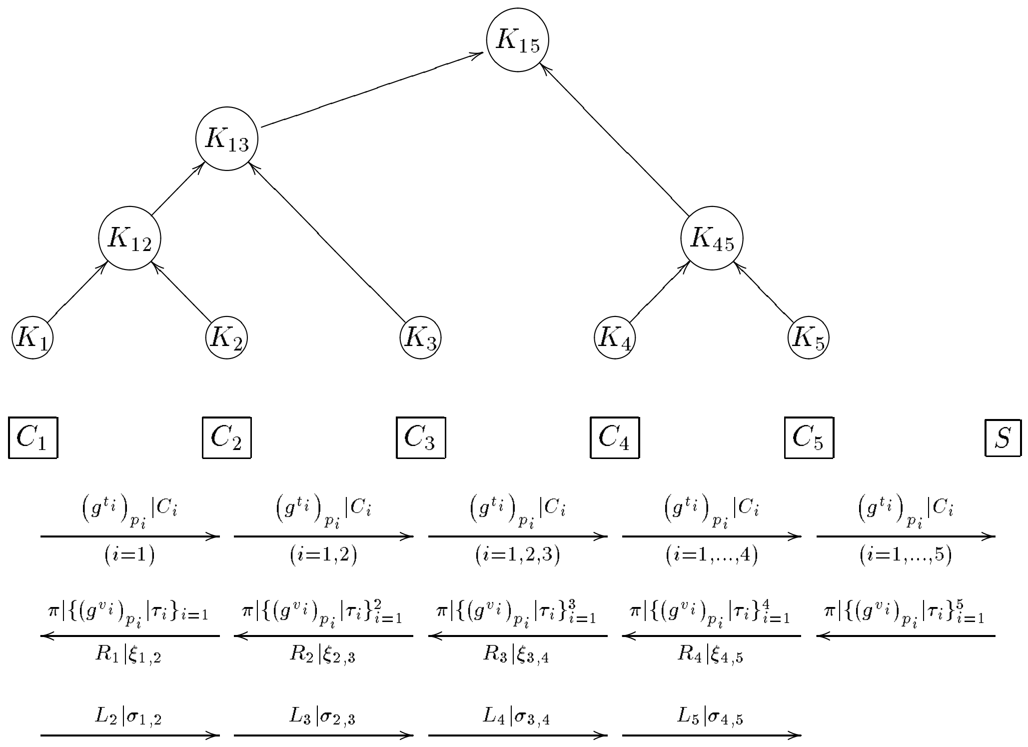

Fig. 2. An example of the protocol with 5 nodes

Flow 1. Upon input[3] M, Client C_i takes $t_i \leftarrow \mathbb{Z}_q$, computes ciphertext $\zeta_i = \left(g^{t_i}\right)_{p_i}$ with his password p_i. Then he sends $\boxed{M|\zeta_i|C_i}$ to C_{i+1} for $i < n$ and to S for $i = n$.

Flow 2. *Server* S. Upon input M, he parses $M = \{\zeta_j|C_j\}_{j=1}^n$. If $n > n_0$, he simply rejects. Otherwise, he decrypts ζ_i to α_i, takes $v_i \leftarrow \mathbb{Z}_q$ and computes $K_i = \alpha_i^{v_i}$, $K_i' = g^{|K_i|_p}, K_{si} = f(K_i)$, for $i = 1, \cdots, n$. Then he prepares and sends $\boxed{\pi|\{\eta_j|\tau_j\}_{j=1}^n}$ to C_n, where $\pi = K_1'|C_1|\cdots|K_n'|C_n$, $\tau_j = F_{K_{sj}}(\pi), \eta_j = (g^{v_j})_{p_j}$.

Client C_i. When receiving message M, C_i first parses $M = \pi|\{\eta_j|\tau_j\}_{j=1}^i|R_i|\xi_{i,i+1}$, where by default $R_n|\xi_{n,n+1}$ is an empty string. If $\eta_i = (g^{t_i})_{p_i}$, he rejects[4]; otherwise, he decrypts η_i to β_i, computes $K_i = \beta_i^{t_i}$, and $K_{si} = f(K_i)$. He verifies if K_i' in π is equal to $g^{|K_i|_p}$ and if $\tau_i = F_{K_{si}}(\pi)$. For the case $i < n$, C_i additionally computes $sk_{i,i+1} := f(K_{i,i+1}) = f((K_{i+1}')^{|K_i|_p})$, where K_{i+1}' is extracted from π, and verifies if $\xi_{i,i+1} = F_{sk_{i,i+1}}(R_i|0)$.

If some verification above fails, C_i rejects; otherwise, C_i $(2 \le i \le n)$ begins to prepare the outgoing message (case $i = 1$ will be described in Flow 3). C_i computes R_{i-1} using R_i, K_i and π. By default, $R_n = nil$. He computes $sk_{i-1,i} = f((K_{i-1}')^{|K_i|_p})$, where K_{i-1}' is extracted from π. Finally, he sends $\boxed{\pi|\{\eta_j|\tau_j\}_{j=1}^{i-1}|R_{i-1}|\xi_{i-1,i}}$ to C_{i-1}, where $\xi_{i-1,i} = F_{sk_{i-1,i}}(R_{i-1}|0)$.

Flow 3. Upon receiving M, C_i does the following. For $i = 1$, the verification is done in Flow 2; for $i > 1$, C_i parses $M = L_i|\sigma_{i-1,i}$, verifies if $\sigma_{i-1,i} = F_{sk_{i-1,i}}(L_i|1)$.

If the verification is successful, then C_i accepts and computes the group session key K_{1n} from K_i, R_i, L_i and π, where by default $L_1 = R_n = nil$. Then, the outgoing message is prepared as follows. For $i = n$, no outgoing message is computed. For $i < n$, C_i first computes L_{i+1} using L_i, K_i and π. Then, he computed and sends $\boxed{L_{i+1}|\sigma_{i,i+1}}$ to C_{i+1}, where $\sigma_{i,i+1} = F_{sk_{i,i+1}}(L_{i+1}|1)$.

5 Security Theorem

Now we prove the security of the protocol in Section 4 in the random oracle and ideal cipher model.

Theorem 1. *Let p, q be large primes with $p = 2q+1$; G_q be a subgroup of order q in $\mathbb{Z}_p$; $g \in G_q\backslash\{1\}$; D be the password dictionary of a polynomial size; $f : G_q \to \{0,1\}^c$ be a random oracle; F_k be an existentially unforgeable message authentication code with a secret key $k \in \{0,1\}^c$; $(\cdot)_{p_i}$ be an ideal cipher from G_q to G_q with a password p_i as the secret key. Then under the DDH assumption, the protocol in Section 4 is secure.*

Proof. Assume $Ins = \langle \mathsf{C}_1^{l_1}, \cdots, \mathsf{C}_n^{l_n}, \mathsf{S}^{l_s}\rangle$ are jointly executing the protocol. Let

$$\mathbf{ed}_i^{l_i} = \left(t_i, \tau_i|\pi|\left(g^{v_i}\right)_{p_i}, R_i|\xi_{i,i+1}, L_i|\sigma_{i-1,i}\right)$$

[3] If $i = 1$, $M := nil$ by default

[4] This check is to avoid basing the security on an assumption stronger than DDH.

be the essential data for C_i, where by default $L_1|\sigma_{0,1}$ and $R_n|\xi_{n,n+1}$ are empty strings. Let $\mathbf{ed}_S^{l_s} = \left\{\left(g^{t_i}\right)_{p_i}|C_i, v_i\right\}_{i=1}^{n}$ be the essential data for S. For clarity, we specify: $L_i|\sigma_{i-1,i}$ originates from $\mathsf{C}_{i-1}^{l_{i-1}}$, $(g^{t_i})_{p_i}|C_i$ originates from $\mathsf{C}_i^{l_i}$, $R_i|\xi_{i,i+1}$ originates from $\mathsf{C}_{i+1}^{l_{i+1}}$, and $\tau_i|\pi|(g^{v_i})_{p_i}$ originates from S^{l_s}.

Now we show the completeness. First of all, the uniqueness of $\mathbf{ed}_i^{l_i}$ (resp. $\mathbf{ed}_S^{l_s}$) is violated with a negligible probability only, as t_i (resp. $v_1, \cdots, v_n$) is random within party C_i (resp. S). From Lemma 2, K_{1n} can be determined using $\mathbf{ed}_i^{l_i}$, as $K_i = (g^{v_i})^{t_i}$ is known to C_i.

Assume partnered instances $\mathsf{C}_i^{l_i}$ and $\mathsf{C}_j^{l_j}$ (w.r.t. an *uncorrupted Ins*) accept such that $\mathsf{C}_t^{l_t}$ for $t \leq \max(i,j)$, $\mathsf{C}_r^{l_r}$ for $r > \max(i,j)$ and S^{l_s} have a valid view on $\mathbf{ed}_t^{l_t}, \{t_r, \tau_r|\pi|\left(g^{v_r}\right)_{p_r}, R_r|\xi_{r,r+1}\}$, and $\mathbf{ed}_S^{l_s}$, respectively. Thus, $\langle R_i, L_i, \pi\rangle$ (resp. $\langle R_j, L_j, \pi\rangle$) seen by $\mathsf{C}_i^{l_i}$ (resp. $\mathsf{C}_j^{l_j}$) is consistent with DH($\mathbf{T}$), where the leaf Exp-Ks of DH($\mathbf{T}$) are $K_1, \cdots, K_n$ for $K_t = g^{t_t v_t}$(known to C_t). Thus, both C_i and C_j can compute $K_{1,n}$ as in DH($\mathbf{T}$).

Now we turn to the authentication and privacy. We regard the protocol execution as a game between a simulator $\mathcal{M}$ and an adversary $\mathcal{A}$. $\mathcal{M}$ first provides $\mathcal{A}$ with parameters p, q, G_q, g, F, E, f, and assigns passwords as in the real world. In addition, $\mathcal{M}$ maintains the random oracle f and ideal cipher $()_{pw}$ as follows.

Random Oracle f. $\mathcal{M}$ maintains a list L_f. Upon a query X, if X is not recorded in L_f, it replies with $\rho \leftarrow \{0,1\}^c$ and then records (X, ρ) into L_f; otherwise, the answer is ρ in an existing record (X, ρ).

Simulating **Ideal Cipher** $(\cdot)_{pw}$: $\mathcal{M}$ maintains a list L_{pw} which is initially empty.

- Encryption query $(encr, pw, \alpha)$. If α is not recorded in L_{pw}, he takes $\zeta \leftarrow G_q$ and adds (α, ζ, b) to L_{pw}, where $b = 1$ if $\mathcal{M}$ issues the query w.r.t. a corrupted client or $\mathcal{A}$ issues the query; $b = 0$ if $\mathcal{M}$ issues the query w.r.t. an uncorrupted client.[5] Then $\mathcal{M}$ returns ζ. On the other hand, if $(\alpha, \zeta, *) \in L_{pw}$ for some ζ, he simply returns ζ.
- Decryption query $(decr, pw, \zeta)$. If ζ is not recorded in L_{pw}, he takes $x \leftarrow \mathbb{Z}_q, X = g^x$ and adds $(X, \zeta, 0)$ to L_{pw}. Then he returns X. Otherwise, he returns X from an existing (X, ζ, b).

Let the real protocol execution be Γ and the game between $\mathcal{M}$ and $\mathcal{A}$ be Γ_0. The view of $\mathcal{A}$ in Γ and Γ_0 differs only if the ideal cipher oracle maintained by $\mathcal{M}$ breaks the permutation property. Thus, an easy calculation leads to the following lemma.

Lemma 3. $\mathbf{Adv}(\mathcal{A}, \Gamma) \leq \mathbf{Adv}(\mathcal{A}, \Gamma_0) + \frac{q_I^2}{q}$, *where q_I is the total number of ideal cipher queries.*

We then construct a sequence of modified games $\Gamma_1, \Gamma_2, \cdots, \Gamma_\nu$ from Γ_0, and show that

$$\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_{i-1})] \leq \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_i)] + \mathbf{negl}(\kappa), 1 \leq i \leq \nu. \tag{2}$$

Finally, we show $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_\nu)]$ is upper bounded by $\frac{1}{2} + \frac{n_0 Q_{send}}{2|D|} + \mathbf{negl}(\kappa)$. Note in order for our proof to be more intelligible, we do not bother to give the concrete bound $\mathbf{negl}(\kappa)$ although it is not difficult to do so.

Game Γ_1. We modify Γ_0 to Γ_1 such that in $\mathbf{Execute}(\mathsf{C}_1^{l_1}, \cdots, \mathsf{C}_n^{l_n}, S^{l_s})$, K_i is random in G_q (instead of $(g^{t_i})^{v_i}$) for any uncorrupted C_i. We have the following lemma.

[5] Note this specification is well defined. Indeed, during response to every query (e.g. **Execute**, **Send**, etc.) if an encryption query is issued by $\mathcal{M}$, pw is always clearly associated with some specific client C_i.

Lemma 4. $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_1)] = \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_0)] + \mathbf{negl}(\kappa)$.

Proof. If the conclusion were wrong, we construct an adversary $\mathcal{B}_1$ to solve the DDH problem. Upon input (α, β, γ), he simulates Γ_1 against $\mathcal{A}$, except for a **Execute** query. In this case, the simulation is normal, except for any uncorrupted C_i, (g^{t_i}, g^{v_i}, K_i) is taken as a random output of $Rud_1(\alpha, \beta, \gamma)$ (see Appendix 1 for details). Finally, $\mathcal{B}_1$ outputs 1 if $\mathcal{A}$ succeeds; otherwise, it outputs 0 or 1 randomly. By property of Rud_1 algorithm, the adversary view in the simulated game is distributed as in Γ_0 if (α, β, γ) is a DH tuple; otherwise, it is distributed as in Γ_1. Thus, $\mathbf{Adv}(\mathcal{B}_1)$ $=\frac{1}{2}|\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_1)] - \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_0)]|$, non-negligible, contradicting the DDH assumption. □

Game Γ_2. Now we define Γ_2 to be Γ_1, except for the **Execute** oracle with *no corrupted client*. Recall in Γ_1, the transcript of this query is $\{(g^{t_i})_{p_i}|C_i\}_1^n, \{(g^{v_i})_{p_i}\}_1^n, \{R_i\}_1^{n-1}, \{L_i\}_2^n, \{F_{K_{si}}(\pi)\}_1^n$, $\{F_{sk_{i,i+1}}(R_i|0)\}_1^{n-1}, \{F_{sk_{i,i+1}}(L_{i+1}|1)\}_1^{n-1}$, where R_i, L_i, π are extracted from DH(T), $K_{si}, sk_{i,i+1}$ are consistent with DH(T) (i.e., $K_{si} = f(K_i)$ for $K_i' \in$ DH($\mathbf{T}$)$; sk_{i,i+1} = f(K_{i,i+1})$ such that $K_{i,i+1}' \in$ DH(T) for $v_{i,i+1} \in \mathbf{T}$ or $K_{i,i+1} \in$ DH($\mathbf{T}$) for $v_{i,i+1} \notin \mathbf{T}$). In Γ_2, this oracle is modified such that DH($\mathbf{T}$) is replaced by R($\mathbf{T}$) and $K_{si}, sk_{i,i+1}$ are consistent with R($\mathbf{T}$).

Lemma 5. $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_2)] = \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_1)] + \mathbf{negl}(\kappa)$.

Proof. If this is not true, we build an adversary $\mathcal{B}_2$ to distinguish DH($\mathbf{T}$) and R($\mathbf{T}$). Let ν_2 be an upper bound for $\sharp$**Execute** queries with no corrupted client. Upon input MH($\mathbf{T}$), which is either DH($\mathbf{T}$) or R($\mathbf{T}$), $\mathcal{B}_2$ does the following. He takes l randomly from $\{1, \cdots, \nu_2\}$, and simulates Γ_1 with $\mathcal{A}$ against it, except for **Execute** queries with no corrupted client. In this case, the first $(l-1)$ such queries are answered according to Γ_2. The oracle for the l-th such query is according to Γ_1, except that DH($\mathbf{T}$) is replaced by MH($\mathbf{T}$) and that each $sk_{r,r+1}$ for $v_{r,r+1} \in \mathbf{T}$ and each K_{si} are taken randomly from $\{0,1\}^c$ without querying L_f. In order of consistency, the random oracle f is modified as follows. Upon a query x, he first verifies if $g^{|x|_p}$ is the entry in MH($\mathbf{T}$) for some leaf $v_i \in \mathbf{T}$ or at some node $v_{t,t+1} \in \mathbf{T}$. If yes, the answer is K_{si} for the former and $sk_{t,t+1}$ for the latter, where both K_{si} and $sk_{t,t+1}$ are defined in the l-th **Execute** oracle with no corrupted client. If no, the answer is normal. The remaining **Execute** queries with no client corrupted are answered according to Γ_1. Finally, $\mathcal{B}_2$ outputs 1 if $\mathcal{A}$ succeeds; otherwise, it outputs 0 or 1 equally likely.

Let $\Gamma_1^{(o)}$ denote the variant of Γ_1 such that the first o **Execute** oracles with no corrupted client are according to Γ_2 while the rest of such queries are according to Γ_1. Thus, $\Gamma_1^{(0)} = \Gamma_1$ and $\Gamma_1^{(\nu_2)} = \Gamma_2$. Note if MH($\mathbf{T}$) is DH($\mathbf{T}$), the simulated game is distributed as $\Gamma_1^{(l-1)}$; otherwise, it is distributed as $\Gamma_1^{(l)}$. Thus, $\mathbf{Adv}(\mathcal{B}_2) = \frac{1}{2\nu}|\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_1)] - \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_2)]|$, non-negligible, contradicting Lemma 1. □

Game Γ_3. Γ_3 is the same as Γ_2, except for the query $\mathbf{Send}(1, S, l_s, n, \zeta_1|C_1|\cdots|\zeta_n|C_n)$ with $(*, \zeta_i, 1) \in L_{p_i}$ for some uncorrupted C_i, or the query $\mathbf{Send}(2, i, l_i, j, \pi|\{\eta_j|\tau_j\}_1^i|R_i|\xi_{i,i+1})$ with $(*, \eta_i, 1) \in L_{p_i}$. Upon any of these *exception* events, we directly announce the success of $\mathcal{A}$. Intuitively, such special events are possible when adversary guesses p_i correctly.

Lemma 6. $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_2)] \leq \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_3)]$.

Proof. Before an exception event in **Send** occur, the adversary view in Γ_2 and Γ_3 is identically distributed. On the other hand, once it occurs, $\mathcal{A}$ already succeeds in Γ_3. □

Game Γ_4. Γ_4 is defined to be Γ_3, except for the following. Recall in Γ_3, K_i in $\mathbf{Send}(1, S, l_s, n, M)$ is set as $(g^{t_i})^{v_i}$. In Γ_4, this is modified such that K_i for an *uncorrupted* C_i is taken randomly from G_q and K_i for a corrupted C_i is still normally $(g^{t_i})^{v_i}$. In the end, the simulator stores a record

$$< \zeta_1|\eta_1|K_1|C_1, \cdots, \zeta_n|\eta_n|K_n|C_n; \{K_s : v_s \in \mathbf{T}, \text{ or } v_s = v_{t,t+1} \notin \mathbf{T} \text{ for } t < n\} >$$

into a list Ω for the future reference.

Recall in Γ_3, in $\mathbf{Send}(2, i, l_i, j, M)$, C_i computes K_i as $(g^{v_i})^{t_i}$. However, in Γ_4, this is no longer consistent as K_i for uncorrupted C_i is taken random in $\mathbf{Send}(1, S, l_s, n, M)$. To overcome this problem, the simulator searches for a record W in Ω such that $(\zeta_i|\eta_i|K_i|C_i)$ is the i-th component of W for some K_i. If such a record does not exist or if there are multiple such records, it rejects; otherwise, let K_i be defined this way and the remaining simulation in this oracle is normal.

$\mathbf{Send}(3, i, l_i, j, M)$ is normal (recall K_i, $sk_{i-1,i}$ and $sk_{i,i+1}$ already exist after $\mathbf{Send}(2, i, l_i, *)$).

Lemma 7. $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_4)] = \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_3)] + \mathbf{negl}(\kappa)$.

Proof. In order to relate Γ_4 and Γ_3, we define a game Γ_4'. Γ_4' follows Γ_4, except that K_i in $\mathbf{Send}(1, S, l_s, n, M)$ for an uncorrupted C_i is set as $(g^{t_i})^{v_i}$ (instead of being random from G_q). We first claim $|\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_3)] - \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_4')]|$ is negligible. By definition of Γ_4', the adversary view in Γ_4' differs from Γ_3 only in the $\mathbf{Send}(2, i, l_i, j, M)$ oracle where the record W is not found or multiple records are found while the normal verification w.r.t Γ_3 succeeds. Note a multi-record event happens negligibly only. To show the claim, it suffices to prove such an event in the case W is not found (denoted by $\mathbf{bad\_rej}$), happens negligibly in Γ_4'. Toward this goal, we modify the simulator of Γ_4' slightly: (1) whenever it computes g^{t_i} or g^{v_i} for an uncorrupted C_i, it records (t_i, g^{t_i}) and (v_i, g^{v_i}) into a list Q, (2) whenever he simulates the plaintext $X = g^x$ for an unrecorded ideal cipher query $(decr, p_i, Y)$, it records (x, X) into Q. Note item (1) is well defined as the simulator knows t_i (resp. v_i) by the protocol specification, and item (2) is well defined by the simulation code for the ideal cipher. Under the revised simulation, the adversary view will not change. Whereas, for any record $(A, B, 0) \in L_{pw}$, $(*, A)$ must be recorded in Q.

Now if $\Pr[\mathbf{bad\_rej}(\Gamma_4')]$ is non-negligible, a DDH breaker $\mathcal{B}_4$ can be built as follows. Let ν be the upper bound of $\sharp\mathbf{Send}(1, i, l_i, j, M)$ queries and μ be the upper bound of $\sharp$ideal cipher queries recorded with indicating bit $b = 0$. Upon input (α, β, γ), $\mathcal{B}_4$ takes $z_1 \leftarrow \{1, \cdots, \nu\}$ and $z_2 \leftarrow \{1, \cdots, \mu\}$ randomly. Then he simulates Γ_4', except

1. when the z_1-th $\mathbf{Send}(1, i, l_i, j, M)$ query occurs. In this case, the oracle simulates the output using $\alpha_i = \alpha$. As the exponent t_i of α is unknown, he does not record a pair (t_i, α_i) into Q.
2. upon the z_2-th unrecorded ideal cipher query with indicating bit $b = 0$. If the query is $(decr, pw, \zeta)$, he defines the plaintext to be β. If the query is $(encr, pw, X)$, it must be issued by the simulator either in computing ζ_i for an uncorrupted C_i or at S in computing η_j for an uncorrupted C_j. In the former case, he re-defines $\alpha_i = \beta$. In the latter case, he first re-defines $\beta_j = \beta$, then let $K_j = \gamma$ if ζ_j is from the z_1-th $\mathbf{Send}(1, *, *, *)$ query; let $K_j = \beta_j^{t_j}$ otherwise, for $(g^{t_j}, \zeta_j, b) \in L_{p_j}$ (note (t_j, g^{t_j}) is recorded in Q: it is true in $\mathbf{Execute}$ query as C_j is uncorrupted and ζ_j is computed by simulator; it is true in $\mathbf{Send}(2, S, l_s, n, M)$, as in case of $(*, \zeta_j, 1) \in L_{p_j}$ $\mathcal{A}$ is announced success before computing K_j). Since the exponent of β is unknown, β is not recorded into Q.

Use $\mathbf{bad\_rej}(z_1, z_2)$ to denote the event: a $\mathbf{Send}(2, i, l_i, j, M)$ query is issued while (1) $\mathbf{Send}(1, i, l_i, *, *)$ is the z_1-th $\mathbf{Send}(1, *, *, *, *)$ query and (2) ζ_i in M is from the z_2-th unrecorded ideal cipher query with $b = 0$, (3) exception queries of item 1 and item 2 in the simulation do not coincide. In case of $\mathbf{bad\_rej}(z_1, z_2)$ event, $\mathcal{B}_4$ outputs 1; otherwise, it outputs 0 or 1 randomly.

If (α, β, γ) is a DH tuple, the adversary view in the simulated game is perfectly consistent with Γ_4'. Thus, $\Pr[\mathcal{B}_4(\mathrm{DH})] = \frac{\Pr[\mathbf{bad\_rej}]}{2\nu\mu} + \frac{1}{2}$ (note if verification of whether $\eta_i = \zeta_i$ in $\mathbf{Send}(2, i, l_i, j, M)$ will guarantee that a violation of (3) in $\mathbf{bad\_rej}(z_1, z_2)$ is no longer a $\mathbf{bad\_rej}$ event). If a $\mathbf{bad\_rej}(z_1, z_2)$ happens to some $\mathbf{Send}(2, i, l_i, M)$ query, then the simulation of letting $K_i = \gamma$ in item 2 never occurs (otherwise, the record W for $\mathbf{Send}(2, i, l_i, M)$ exists). Thus, if (α, β, γ) is random, $\mathbf{bad\_rej}(z_1, z_2)$ event implies that γ is independent of the adversary view. However, the $\mathbf{bad\_rej}(z_1, z_2)$ also implies K_i' in π is verified to be $g^{|\gamma|_p}$, negligible! Thus, $\mathbf{Adv}(\mathcal{B}_4)$ is non-negligible, contradicting the DDH assumption.

Now we show that $|\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_4')] - \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_4)]|$ is negligible. Note that the difference in Γ_4 and Γ_4' is only that K_i is taken normally or randomly. If the success gap is non-negligible, a simple hybrid argument can be used to break the DDH assumption. □

Game Γ_5. Now we modify Γ_4 to Γ_5 with the only difference in $\mathbf{Send}(2, i, l_i, M)$ oracle. In this case, when the unique record W is found, the verification of τ_i will additionally check if π is consistent with W (i.e., checking if each K_t' in π is equal to $g^{|K_t|_p}$ for K_t in W).

Lemma 8. $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_5)] = \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_4)] + \mathbf{negl}(\kappa)$.

Proof. If there were wrong, then the probability *prob* of event **Inc**: $\mathbf{Send}(2, i, l_i, j, M)$ oracle in Γ_4 accepts while π is inconsistent with the underlying W, is non-negligible. We construct an adversary $\mathcal{B}_5$ to break the security of F. Let ν be the upper bound of the number of records in Ω. $\mathcal{B}_5$ chooses z randomly from $\{1, \cdots, \nu\}$, $t \leftarrow \{1, \cdots, n_0\}$, where n_0 is the maximum number of clients allowed in one protocol execution. $\mathcal{B}_5$ simulates Γ_4 with $\mathcal{A}$ against it, except for the $\mathbf{Send}(1, S, l_s, n, M)$ oracle recording the z-th record $\hat{W}$ in Ω. In this case, if C_t is corrupted (so no **Inc** event happens to C_t), this simulation is normal. If C_t is uncorrupted, the simulation is normal except that K_{st} is defined as the secret key hidden in his challenger's F. To be consistent, he modifies the random oracle f: if the input is K_t in $\hat{W}$, he aborts with failure; otherwise, the answer is normal. Later upon query $\mathbf{Send}(2, t, l_t, j, M)$ with the record being $\hat{W}$, he checks if π is consistent with $\hat{W}$. In case of yes, the simulation is normal (with the help of the F challenger to compute $F_{K_{si}}(\pi)$); otherwise, it outputs (τ_t, π) as his forgery, where τ_t, π is extracted from M.

We first show that if K_t in $\hat{W}$ with uncorrupted C_t is issued to L_f non-negligibly, then a discrete log solver $\mathcal{D}_5$ can be built. Upon input $\gamma \in G_q$, $\mathcal{D}_5$ plays the role of $\mathcal{B}_5$ and the challenger of $\mathcal{B}_5$, except the query $\mathbf{Send}(1, S, l_s, n, M)$ toward the z-th record $\hat{W}$. In this case, if C_t is corrupted, the simulation is normal; otherwise, $K_{st} \leftarrow G_q$, K_t is left undefined and K_t' is defined to γ. The rest of the simulation is normal until a random oracle query x with $g^{|x|_p} = \gamma$ occurs. In this case, it outputs $|x|_p$ as the discrete log of γ. Note the simulation by $\mathcal{D}_5$ is perfectly consistent with that by $\mathcal{B}_5$. Thus, the discrete log is solved non-negligibly, contradiction.

We come back to analyze the game by $\mathcal{B}_5$. Assume K_t in $\hat{W}$ for the uncorrupted C_t case is never issued to L_f. Then, the adversary view is distributed according to Γ_4, unless $\mathbf{Send}(2, t, l_t, j, M)$ with record $\hat{W}$ occurs such that π is inconsistent with $\hat{W}$. In this case, an **Inc** event happens to $\mathbf{Send}(2, t, l_t, j, M)$ if and only it (π, τ_i) constitutes an F-forgery. Thus, $\Pr[\mathbf{Succ}(\mathcal{B}_5)] = \frac{1}{n_0\nu} \cdot prob + \mathbf{negl}(\kappa)$, non-negligible, contradicting the security of F. □

Game Γ_6. Γ_6 is defined to Γ_5, except for the following settings, where no client is *corrupted.*

- When oracle $\mathbf{Send}(2, i, l_i, j, M)$ verifies $\xi_{i,i+1}$, it additionally checks if R_i is consistent with the underlying record W. Furthermore, R_{i-1} is computed directly from W (i.e., for any $K'_s \in R_{i-1}$, $K'_s = g^{|K_s|_p}$, where K_s is from W).
- When oracle $\mathbf{Send}(3, i, l_i, j, M)$ verifies $\sigma_{i-1,i}$, it additionally verifies if L_i is consistent with the underlying record W. Furthermore, L_{i+1} is computed directly from W (i.e., for any $K'_s \in L_{i+1}$, $K'_s = g^{|K_s|_p}$, where K_s is from W).

Lemma 9. $\Pr[\mathbf{Succ}(\mathcal{A}, \varGamma_6)] = \Pr[\mathbf{Succ}(\mathcal{A}, \varGamma_5)] + \mathbf{negl}(\kappa)$.

Proof. In $\varGamma_5$, R_{i-1} is computed from K_i, R_i and π, where K_i is extracted from the underlying record W. In addition, before $\mathbf{Send}(2, i, l_i, j, M)$ computes R_{i-1}, the consistency of π with W is verified. So, the consistency of R_i with W implies the consistency of R_{i-1} with W. Similar argument holds for L_{i+1}. Thus, adversary view in $\varGamma_6$ differs from $\varGamma_5$ only when $\xi_{i,i+1}$ (resp. $\sigma_{i-1,i}$) is successfully verified while R_i (resp. L_i) is inconsistent with W. We denote such an event **Bad**. To prove the lemma, it suffices to show that $\Pr[\mathbf{Bad}(\varGamma_5)]$ is negligible. If this is not true, we construct an adversary $\mathcal{B}_5$ to break the security of F. Let ν be the upper bound of $\sharp$records in $\varOmega$ containing no corrupted client. $\mathcal{B}_5$ takes $z \leftarrow \{1, \cdots, \nu\}, t \leftarrow \{1, \cdots, n_0 - 1\}$. He simulates $\varGamma_5$, except that $sk_{t,t+1}$ w.r.t. the z-th record (denoted by $\hat{W}$) with no corrupted client is defined to be the key hidden in the challenger's F. To be consistent, the random oracle f is modified: if a query $K_{t,t+1}$ in $\hat{W}$ is issued to L_f, then $\mathcal{B}_5$ aborts with failure; otherwise, the answer is normal. Later whenever it is required to compute F value under this key, it asks for help from his challenger unless one of the followings occurs:

1. $\mathbf{Send}(2, t, l_t, j, M)$ oracle with record $\hat{W}$ is ready to verify $\xi_{t,t+1}$. In this case, if R_t is inconsistent with $\hat{W}$, he outputs $(R_t|0, \xi_{t,t+1})$ (from M) as his forgery; otherwise, the response is normal (with the help of the F challenger to compute $F_{sk_{t,t+1}}(R_t|0)$).
2. $\mathbf{Send}(3, t, l_t, j, M)$ oracle with record $\hat{W}$ is ready to verify $\sigma_{t-1,t}$. In this case, if L_t is inconsistent with $\hat{W}$, he outputs $(L_t|1, \sigma_{t-1,t})$ (from M) as his forgery; otherwise, the simulation is normal (with the help of the F challenger to compute $F_{sk_{t,t+1}}(L_{t+1}|1)$).

If the inconsistent event of case 1 and case 2 never occurs, then $\mathcal{B}_5$ terminates with failure.

If the abortion event upon the random oracle query $K_{t,t+1}$ (in $\hat{W}$) (denoted by event **aro**) is replaced with a normal execution, the adversary view before the special inconsistent event occurs is perfectly consistent with $\varGamma_5$. Notice that the inconsistent event w.r.t. $(t, \hat{W})$ is a **Bad** event if and only if the output of $\mathcal{B}_5$ is a forgery. Thus, $\Pr[\mathbf{Succ}(\mathcal{B}_5)] > \frac{\Pr[\mathbf{Bad}(\varGamma_5)]}{\nu n_0} - \Pr[\mathbf{aro}]$. If $\Pr[\mathbf{aro}]$ is negligible, then $\Pr[\mathbf{Succ}(\mathcal{B}_5)]$ is non-negligible, contradicting the security of F.

Now if $\Pr[\mathbf{aro}]$ is non-negligible, then a DDH distinguisher $\mathcal{D}_5$ can be built as follows. Given (α, β, γ), $\mathcal{D}_5$ plays the roles of $\mathcal{B}_5$ and the challenger of $\mathcal{B}_5$, except for oracle $\mathbf{Send}(1, S, l_s, n, M)$ that records $\hat{W}$ (the z-th record with no corrupted client). In this case, he defines $K'_t = \alpha, K'_{t+1} = \beta, K_{t,t+1} = \gamma$. As K_t, K_{t+1} is unknown, leave them undefined. Furthermore, he define $K_{st}, K_{s(t+1)}$ to be random from $\{0,1\}^c$. Later he modifies the random oracle f such that if the query x satisfies $g^{|x|_p} = K'_t$ or $g^{|x|_p} = K'_{t+1}$, the answer is K_{st}, $K_{s(t+1)}$ respectively; otherwise, the answer is normal. The rest of simulation does not involve K_t, K_{t+1} in $\hat{W}$. Thus, the simulation according to the code of $\mathcal{B}_5$ is well defined. Finally, if a random oracle query $x = \gamma$ occurs, $\mathcal{D}_5$ outputs 1 and terminates. If such a query never occurs, it outputs 0 or 1 equally likely. If (α, β, γ) is DH tuple, then the adversary view is exactly consistent with the simulation by $\mathcal{B}_5$, $\Pr[\mathcal{D}_5(\mathrm{DH}) = 1] = \frac{1+\Pr[\mathbf{aro}]}{2}$. On the other hand, if the input is a random tuple, then the underlying $\Pr[\mathbf{aro}]$ is negligible; otherwise, the discrete log problem can be solved (specifically, upon input Y, such an adversary just simulates by playing the role of $\mathcal{B}_5$ and the challenger of $\mathcal{B}_5$, except that $K_{t,t+1} \in \hat{W}$ is left undefined but $sk_{t,t+1}$ is taken randomly from $\{0,1\}^c$, and that if $v_{t,t+1} \in \mathbf{T}$, $K'_{t,t+1}$ w.r.t. $\hat{W}$ is defined as Y. The rest of simulation is normal until an random oracle query y with $g^{|y|_p} = Y$ occur). Thus, the advantage of $\mathcal{D}_5$ is non-negligible, contradicting the DDH assumption. □

Game Γ_7. Γ_7 is modified from Γ_6 such that upon $\mathbf{Send}(1, S, l_s, n, M)$ query with no corrupted client, K_s in the record W is taken uniformly from G_q, for each $v_s \in \mathbf{T}$ or $v_s = v_{i,i+1} \notin \mathbf{T}$.

Lemma 10. $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_7)] = \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_6)] + \mathbf{negl}(\kappa)$

Proof. If the lemma is wrong, a distinguisher $\mathcal{B}_7$ for DH($\mathbf{T}$) and R($\mathbf{T}$) can be built as follows. Let ν be the upper bound of ♯records in Ω with no corrupted client. $\mathcal{B}_7$ takes z randomly from $\{1, \cdots, \nu\}$. Upon input MH($\mathbf{T}$), which is either DH($\mathbf{T}$) or R($\mathbf{T}$), $\mathcal{B}_5$ simulates according to Γ_7, except when the z-th $\mathbf{Send}(1, S, l_s, n, M)$ with no client corrupted occurs. In this case, he defines the record (denoted by $\hat{W}$) using MH($\mathbf{T}$). Since K_s for $v_s \in \mathbf{T}(s \neq (1, n))$ is unknown, he describes it as satisfies $g^{|K_s|_p} = \delta'_s$, where δ'_s is the entry in MH($\mathbf{T}$) corresponding to $v_s \in \mathbf{T}$. He take K_{si} ($i = 1, \cdots, n$), $sk_{t,t+1}$ for each $v_{t,t+1} \in \mathbf{T}$ to be uniformly random. In order of consistency, the random oracle f is modified: if an query x satisfies $g^{|x|_p} = \delta'_i$ or $\delta'_{t,t+1}$, then the answer is K_{si} for the former and $sk_{t,t+1}$ for the latter. The rest of the simulation is normal. Finally, if $\mathcal{A}$ succeeds, $\mathcal{B}_7$ outputs 1; otherwise, it outputs 0 or 1 equally likely.

Let $\Gamma_6^{(l)}$ be the game such that the first l $\mathbf{Send}(1, S, l_s, n, M)$ with no corrupted client is simulated according to Γ_7 while the rest of such queries are simulated according to Γ_6. If the input is DH($\mathbf{T}$), the simulated game is distributed according to $\Gamma_7^{(z-1)}$; otherwise, it is distributed according to $\Gamma_7^{(z)}$. So, $\mathbf{Adv}(\mathcal{B}_7) = \frac{|\Pr[\mathbf{Succ}(\mathcal{A},\Gamma_7)] - \Pr[\mathbf{Succ}(\mathcal{A},\Gamma_6)]|}{2\nu}$, non-negligible, contradicting Lemma 1. □

Game Γ_8. Recall that when the simulator computes ζ_i (or η_i) in Γ_7 for an uncorrupted C_i (in either **Execute** query or **Send** query), he first takes α_i (or β_i) randomly from G_q and then obtains ζ_i (or η_i) by issuing an ideal cipher query to L_{p_i}. Now we modify Γ_7 to Γ_8 such that whenever the simulator needs to compute ζ_i (or η_i) with respect to an uncorrupted C_i, it simply takes ζ_i (or η_i) randomly from G_q (i.e., without computing α_i (or β_i) and issuing the ideal cipher query at all). We stress that in Γ_8, the simulator no longer issues an encryption query w.r.t. an uncorrupted client.

Lemma 11. $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_8)] = \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_7)] + \mathbf{negl}(\kappa)$.

Proof. Let $\Gamma_7^{(l)}$ be the variant of Γ_7 such that the first l computations of ζ_i or η_i w.r.t. an uncorrupted client are according to Γ_8 while the rest of such computations are according to Γ_7. If l is upper bounded by ν. Then $\Gamma_7^{(0)} = \Gamma_7$, $\Gamma_7^{(\nu)} = \Gamma_8$. To prove the lemma, it suffices to show that adversary views in any adjacent pair $\Gamma_7^{(l-1)}$ and $\Gamma_7^{(l)}$ are statistically close. Now we simulate $\Gamma_7^{(l-1)}$ and analyze its relationship with $\Gamma_7^{(l)}$. Before the l-th computation of ζ_i or η_i w.r.t an uncorrupted C_i occurs, the adversary view in the simulated game is identically distributed as in $\Gamma_7^{(l)}$. W.L.O.G., assume the l-th such computation is for ζ_i (in either **Execute** or $\mathbf{Send}(1, i, l_i, j, M)$ oracle). In the simulation, we first set $\alpha_i = \tilde{\alpha}_i$ for $\tilde{\alpha}_i \leftarrow G_q$. If $(\tilde{\alpha}_i, \tilde{\zeta}_i, *) \in L_{p_i}$ for some $\tilde{\zeta}_i$, set $\zeta_i = \tilde{\zeta}_i$; otherwise, set $\zeta_i = \tilde{\zeta}_i$ for $\tilde{\zeta}_i \leftarrow G_q$. Notice that in both **Execute** and $\mathbf{Send}(1, i, l_i, j, M)$ oracle, the oracle output is independent of $\tilde{\alpha}_i$ when $\tilde{\alpha}_i$ was unrecorded in L_{p_i} before. Recall in $\Gamma_7^{(l)}$, simply set $\zeta_i = \tilde{\zeta}_i$ for $\tilde{\zeta}_i \leftarrow G_q$. Thus, if $\tilde{\alpha}_i$ was unrecorded, the adversary view in the simulation is identically distributed as in $\Gamma_7^{(l)}$. Denote the event $\tilde{\alpha}_i$ was recorded before as $\mathbf{differ}_l$. Then $\Pr[\mathbf{differ}_l] = \frac{d_i}{q}$, where d_i is the number of the current records in L_{p_i}. In the following, we assume $\neg\mathbf{differ}_l$. Then, this identical distribution with $\Gamma_7^{(l)}$ remains unless $(encr, p_i, \tilde{\alpha}_i)$ query or $(decr, p_i, \tilde{\zeta}_i)$ query occurs, as (1) both the internal states except L_{p_i} and the simulation specification are perfectly consistent with $\Gamma_7^{(l)}$, (2) the output of $(encr, p_i, X)$ query for $X \neq \tilde{\alpha}_i$ (or the output of $(decr, p_i, Y)$ query for $Y \neq \tilde{\zeta}_i$) is identically distributed as in $\Gamma_7^{(l)}$. In the following, assume $(encr, p_i, \tilde{\alpha}_i)$ has not occurred yet. Then, before $(decr, p_i, \tilde{\zeta}_i)$ query, $\tilde{\alpha}_i$ is independent of the adversary view. Thus, for each $(encr, p_i, X)$ query, $X = \tilde{\alpha}_i$ holds with probability at most $\frac{1}{q-m_i}$, where m_i is the current number of records in L_{p_i}. On the other hand, when $(decr, p_i, \tilde{\zeta}_i)$ occurs, the answer is unform in G_q in $\Gamma_7^{(l)}$. Thus, the adversary view differs from $\Gamma_7^{(l)}$ with probability $\frac{d_i}{q}$. Furthermore, if right after $(decr, p_i, \tilde{\zeta}_i)$ query the adversary view in $\Gamma_7^{(l-1)}$ and $\Gamma_7^{(l)}$ are identically distributed, then this identical distribution will remain throughout the simulation as now both the internal states are identical. Thus, the simulated game (i.e., $\Gamma_7^{(l-1)}$) is different from $\Gamma_7^{(l)}$ with a probability at most $\frac{2d_i}{q} + \sum_{m_i=d_i}^{\mu} \frac{1}{q-m_i} < \frac{\mu+2}{q-\mu}$, negligible, where μ is the upper bound on the number of records in L_{p_i}. □

Game Γ_9. Recall in $\mathbf{Send}(1, S, l_s, n, \{\zeta_t|C_t\}_1^n)$ oracle of Γ_8, the simulator will check if $(*, \zeta_t, 1) \in L_{p_t})$ for an uncorrupted C_t. If yes, it announces the success of $\mathcal{A}$; otherwise, it issues $(decr, p_t, \zeta_t)$ query. The similar treatment happens to $\mathbf{Send}(2, i, l_i, j, M)$ oracle, where the checked ciphertext is η_i. We modify Γ_8 to Γ_9 such that no decryption query w.r.t. the uncorrupted client is issued when $(*, \zeta_t, 1) \notin L_{p_t}$ for the former (or when $(*, \eta_i, 1) \notin L_{p_i}$ for the latter). This is well defined as α_t (or β_i) for an uncorrupted client is not used to simulate the output. We stress that in Γ_9, the simulator no longer issues a decryption query w.r.t. an uncorrupted client (especially, recall such a query does not occur in **Execute** oracle in $\Gamma_0 \sim \Gamma_9$).

Lemma 12. $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_9)] = \Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_8)] + \mathbf{negl}(\kappa)$.

Proof. The proof strategy is similar to the previous lemma. Define $\Gamma_8^{(l)}$ to be a variant of Γ_8 such that the first l decryption queries w.r.t. an uncorrupted client (issued by the simulator in either **Execute** or **Send** oracle) are ignored (as in Γ_9) and the remaining such queries are simulated according to Γ_8. To prove the lemma, it suffices to show that the adversary view in $\Gamma_8^{(l-1)}$ and $\Gamma_8^{(l)}$ are statistically close. We simulate $\Gamma_8^{(l-1)}$ and analyze the relationship with $\Gamma_8^{(l)}$. Before the l-th decryption query w.r.t. an uncorrupted client occurs, adversary view in the simulated game is exactly the same as in $\Gamma_8^{(l)}$. Assume the l-th such query is $(decr, p_i, Y)$. If Y was recorded before, then adversary view throughout the whole simulation is exactly the same as in $\Gamma_8^{(l)}$. In the following, we assume that Y is unrecorded. In the simulation, $(decr, p_i, Y)$ is answered by taking the plaintext $X \leftarrow G_q$ while the output of the underlying oracle (either **Execute** or **Send** oracle) involving $(decr, p_i, Y)$ query is independent of X and identically distributed as in $\Gamma_8^{(l)}$. Especially, adversary view in the simulation is independent of X. In addition, the adversary view is and will remain identically distributed as in $\Gamma_8^{(l)}$ unless $(encr, p_i, X)$ query or $(decr, p_i, Y)$ occurs, as (1) the internal state of the simulator except L_{p_i} is identically distributed as in $\Gamma_8^{(l)}$, (2) the output for $(encr, p_i, \lambda)$ query with $\lambda \neq X$ and the output of $(decr, p_i, \upsilon)$ query with $\upsilon \neq Y$ are identically distributed as in $\Gamma_8^{(l)}$. In the following, we assume $(encr, p_i, X)$ has not occurred yet. Then, before $(decr, p_i, Y)$ query occurs, X is independent of adversary view. Thus, for each $(encr, p_i, \lambda)$ query, $\lambda = X$ holds with probability at most $\frac{1}{q-m_i}$, where m_i is the current number of records in L_{p_i}. Once $(decr, p_i, \lambda)$

query occurs, the adversary view will remain identical as in $\Gamma_8^{(l)}$ as their internal states are identically distributed after the simulation of the underlying oracle (either **Execute** or **Send** oracle). Hence, the adversary view in $\Gamma_8^{(l-1)}$ and $\Gamma_8^{(l)}$ differs with probability at most $\sum_{m_i=0}^{\mu} \frac{1}{q-m_i} < \frac{\mu}{q-\mu}$, negligible, where μ is the upper bound of number of records in L_{p_i}. $\square$

Upper bounding $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_9)]$. There are three type of success for $\mathcal{A}$ in Γ_9. Firstly, $\mathcal{A}$ can be announced a success at either $\mathbf{Send}(1, S, l_s, n, M)$ query or $\mathbf{Send}(2, i, l_i, j, M))$ query. Secondly, he can breaking the authentication. Thirdly, he can win the test session. We denote type i success by $\mathbf{Succ}_i$, for $i = 1, 2, 3$.

Lemma 13. *If $\mathcal{A}$ issues at most Q_{send}* **Send** *queries, then* $\Pr[\mathbf{Succ}_1(\mathcal{A}, \Gamma_9)] \leq \frac{n_0 Q_{send}}{|D|} + \mathbf{negl}(\kappa)$.

Proof. The simulation of Γ_9 is independent of passwords for uncorrupted clients, except for the checking of $(*, \zeta_i, 1) \in L_{p_i}$ at $\mathbf{Send}(1, S, l_s, M)$, or the checking of $(*, \eta_i, 1) \in L_{p_i}$ at $\mathbf{Send}(2, i, l_i, M)$. Thus, we can separate the simulator into $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\mathcal{M}_2$ holds the passwords of uncorrupted clients and $\mathcal{M}_1$ does the real simulation. Whenever $\mathcal{M}_1$ needs to check if $(*, Y, 1) \in L_{p_i}$ for some uncorrupted C_i, he searches through all possible pw such that $(*, Y, 1) \in L_{pw}$. If such a password does not exist, he decides $(*, Y, 1) \notin L_{p_i}$ by himself; otherwise, this set of passwords by $W(Y)$. He then asks $\mathcal{M}_2$ whether $p_i \in W(Y)$, from which $\mathcal{M}_1$ decides if $(*, Y, 1) \in L_{p_i}$. It is not difficult to see adversary view in this simulation is perfectly consistent with Γ_9. Thus, an easy calculation shows that if $Y_1, \cdots, Y_l$ are the checks issued to $\mathcal{M}_2$ by $\mathcal{M}_1$ w.r.t. C_i, then $\mathbf{Succ}_1$ happens to C_i within the first l checks with probability at most $\sum_{t=1}^{l} \prod_{c=1}^{t} [1 - \frac{1}{|D|-|W(Y_1)|-\cdots-|W(Y_{c-1})|}] \frac{1}{|D|-|W(Y_1)|-\cdots-|W(Y_c)|}$. If each $|W(Y_i)| = 1$, each summation term is exactly $\frac{1}{|D|}$. Thus, the summation gives to $\frac{l}{|D|}$. In addition, one **Send** query contains at most n_0 clients. It follows that $\Pr[\mathbf{Succ}_1(\mathcal{A}, \Gamma_9)] \leq \frac{n_0 Q_{send}}{|D|}$. On the other hand, some $|W(Y_i)| > 1$ happens negligibly only (as two unrecorded encryption queries results in an identical ciphertext Y_i with a negligible probability). Thus, the conclusion follows. $\square$

In the following, we focus on $\Pr[\mathbf{Succ}_2(\mathcal{A}, \Gamma_9)|\overline{\mathbf{Succ}_1}]$ and $\Pr[\mathbf{Succ}_3(\mathcal{A}, \Gamma_9)|\overline{\mathbf{Succ}_1} \wedge \overline{\mathbf{Succ}_2}]$.

Claim 1. If no involved clients are corrupted, the following statements are true, except for a negligible probability.

1. Assume $\tau_i|\pi$ is accepted by $\mathbf{Send}(2, i, l_i, i+1, M)$ w.r.t a record $\hat{W}$. Then, the instance recording $\hat{W}$ indeed computed $\tau_i|\pi$. In addition, if any S^{l_s}, recording $\tilde{W}$, computed $\tau_i|\pi$, then $\tilde{W} = \hat{W}$.
2. If $\xi_{i,i+1}|R_i$ is accepted by $\mathbf{Send}(2, i, l_i, i+1, M)$ w.r.t. $\hat{W}$, then $\exists\ \mathsf{C}_{i+1}^{l_{i+1}}$ referring to $\hat{W}$ has computed $\xi_{i,i+1}|R_i$. In addition, if any $\mathsf{C}_{i+1}^{l'_{i+1}}$ computed $\xi_{i,i+1}|R_i$, then it refers to $\hat{W}$.
3. If $\sigma_{i-1,i}|L_i$ is accepted by $\mathbf{Send}(3, i, l_i, i-1, M')$ w.r.t. $\hat{W}$, then $\exists\ \mathsf{C}_{i-1}^{l_{i-1}}$ referring to $\hat{W}$ has computed $\sigma_{i-1,i}|L_i$. In addition, if any $\mathsf{C}_{i-1}^{l'_{i-1}}$ computed $\sigma_{i-1,i}|L_i$, then it refers to $\hat{W}$.

Proof. 1. $\tau_i|\pi$ is accepted only if π is consistent with $\hat{W}$ and $\tau_i = F_{K_{si}}(\pi)$, where $K_{si} = f(K_i)$ and K_i is in $\hat{W}$. Thus, the instance recording $\hat{W}$ must have computed $\pi|\tau_i$. In addition, if $\pi|R_i$ originates from a $\mathbf{Send}(1, S, l_s, n, M)$, where $\tilde{W}$ is recorded, then K_t in $\hat{W}$ equals K_t in $\tilde{W}$ for $t = 1, \cdots, n$. This happens negligibly if $\tilde{W} \neq \hat{W}$.
2. Verifying $\xi_{i,i+1}$ is processed only if the record $\hat{W}$ is unique and $\tau_i|\pi$ has been accepted.
Existence of $\mathsf{C}_{i+1}^{l_{i+1}}$. If non-existence of a $\mathsf{C}_{i+1}^{l_{i+1}}$ (denoted by event **nex**) happens non-negligibly, we build an attacker $\mathcal{F}$ to break F. Let μ_9 be the upper bound of $\sharp\mathbf{Send}(1, S, l_s, n, M)$ queries with

no client corrupted. $\mathcal{F}$ takes $t \leftarrow \{1, \cdots, \mu_9\}$ and simulates Γ_9, unless the t-th $\mathbf{Send}(1, S, l_s, n, M)$ query with no client corrupted. In this case, the simulation is normal with the record $\hat{W}$. Then, he takes $r \leftarrow \{1, \cdots, n-1\}$, and sets $sk_{r,r+1}$ w.r.t. $\hat{W}$ as the hidden key in F. For consistency, he modifies L_f as follow: upon a query $K_{r,r+1}$ in $\hat{W}$, he aborts with failure; otherwise, the answer is normal. The remaining simulation is normal (with the help of the F challenger), until some $\tau_r|\pi$ has been accepted by a $\mathsf{C}_r^{l_r}$ w.r.t. $\hat{W}$ while no $\mathsf{C}_{r+1}^{l_{r+1}}$ has computed $\xi_{r,r+1}|R_r$ w.r.t $\hat{W}$. In this case, $\mathcal{F}$ announces $\xi_{r,r+1}|R_r$ as his forgery. If such an event never occur, then $\mathcal{F}$ terminates with failure.

Note that if $K_{r,r+1}$ in $\hat{W}$ is never issued to L_f throughout the simulation, then the simulation by $\mathcal{F}$ is perfectly consistent with Γ_9. In this case, a $\mathbf{nex}$ event w.r.t (t, r) happens if and only if the output of $\mathcal{F}$ is a successful forgery. Thus, $\Pr[\mathbf{Succ}(\mathcal{F})] = \frac{1}{\mu_9}\sum_t \frac{1}{n}\sum_r \Pr[\mathbf{nex} \text{ w.r.t. } (t,r)] \geq \frac{\Pr[\mathbf{nex}]}{\mu_9 n_0}$. On the other hand, we show that the query $K_{r,r+1}$ in $\hat{W}$ is issued to L_f with a negligible probability. In case of $v_{r,r+1} \notin \mathbf{T}$, this conclusion holds since $K_{r,r+1}$ is uniform in G_q, independent of the view by $\mathcal{A}$. The case $v_{r,r+1} \in \mathbf{T}$ holds too; otherwise, the discrete log problem can be solved by defining $K'_{r,r+1}$ to Y and running a normal simulation until a random oracle query X with $g^{|X|_p} = Y$ to occur. In this case, it outputs $|X|_p$ as the discrete log of Y. In summary, $\Pr[\mathbf{Succ}(\mathcal{F})]$ is non-negligible, a contradiction to the security of F.

$\hat{W}$ is referred to by $\mathsf{C}_{i+1}^{l'_{i+1}}$. Note $\xi_{i,i+1}|R_i$ can be directly obtained from $\{K_s : v_s \in \mathbf{T}\}$ in W. Let Exp-K$(\mathbf{T}_n) = \{K_v : v \in \mathbf{T}_n\}$, where $\mathbf{T}_n$ is the regular tree with n leaves. Let μ_5 be the upper bound of $\sharp\mathbf{Send}(1, S, l_s, n, M)$ with no corrupted client. Sample μ_5 copies of Exp-K$(\mathbf{T}_n)$, for each $n = 2, \cdots, n_0$. Then if the conclusion is incorrect, then $\xi_{i,i+1}(a, n)|R_i(a, n)) = \xi_{j,j+1}(b, n_2)|R_j(b, n_2)$ holds non-negligibly for some $(i, a, n_1) \neq (j, b, n_2)$, where $\xi_{r,r+1}(t, n)|R_r(t, n)$ is the specific $\xi_{r,r+1}|R_r$, computed from the t-th sample of Exp-K$(\mathbf{T}_n)$. Denote this probability by *prob*. Then we can break F as follows. We first sample Exp-K$(\mathbf{T}_n)$, for each $n = 2, \cdots, n_0$. Then we take i, j, n_1, n_2, a, b uniformly random in their ranges such that $(i, n_1, a) \neq (j, n_2, b)$. If $K_{i,i+1}(a, n_1) = K_{j,j+1}(b, n_2)$, we terminate with failure; otherwise, set $sk_{i,i+1}(a, n_1)$ as the hidden key in F and $sk_{j,j+1}(b, n_2)$ randomly from $\{0,1\}^c$. Then if $R_i(a, n_1) = R_j(b, n_2)$, we output $\xi_{j,j+1}(b, n_2)$ as a forgery of F on message $R_i(a, n_1)|0$. Thus, the attack are successful if (i, n_1, a) and (j, n_2, b) happen to form a collision. Since $K_{i,i+1}(a, n_1) = K_{j,j+1}(b, n_2)$ holds negligibly, we succeed with probability negligibly close to $\frac{prob}{n_0^4 \mu_5^2}$, non-negligible, contradicting the security of F.

3. The proof here is very similar to item 2. We omit it. □

Claim 2. $\Pr[\mathbf{Succ}_2(\mathcal{A}, \Gamma_9)|\overline{\mathbf{Succ}_1}] = \mathbf{negl}(\kappa)$.

Proof. Let $\mathsf{C}_i^{l_i}$ completes and accepts with uncorrupted $\mathbf{pid}_i^{l_i} = \langle C_1, \cdots, C_n, S\rangle$. We first consider the existence of Ins. For simplicity, assume item 1, 2, and 3 in Claim 1 are never violated. So, $\xi_{i,i+1}|R_i$ (for $i < n$) originates from a $\mathsf{C}_{i+1}^{l_{i+1}}$, and $\mathsf{C}_i^{l_i}$ and $\mathsf{C}_{i+1}^{l_{i+1}}$ refer to an identical W. As $\mathsf{C}_{i+1}^{l_{i+1}}$ sent out $\xi_{i,i+1}|R_i$, it must have accepted in Flow 2. Thus, for $j = i+1, \cdots, n-1$, we iteratively have that $\xi_{j,j+1}|R_j$ originates from a $\mathsf{C}_{j+1}^{l_{j+1}}$, and $\mathsf{C}_{j+1}^{l_{j+1}}$ refers to W. On the other hand, $\mathsf{C}_i^{l_i}$ accepts in Flow 3. Thus, for $j = i, \cdots, 2$, we again iteratively have that $\sigma_j|L_j$ originates from a $\mathsf{C}_{j-1}^{l_{j-1}}$, and $\mathsf{C}_{j-1}^{l_{j-1}}$ refers to W. From item 1 in Claim 1, $\pi|\tau_i$ originates from a S^{l_s}, and S^{l_s} computed W. We show $Ins = \langle \mathsf{C}_1^{l_1}, \cdots, \mathsf{C}_n^{l_n}, \mathsf{S}^{l_s}\rangle$ is the required instance list. Indeed, as $\zeta_j|C_j$ is in the j-th component of W (referred by $\mathsf{C}_j^{l_j}$), S^{l_s} has a valid view on $\mathbf{ed}_S^{l_s}$. $\xi_{j,j+1}|R_j$ for $j < i$ originates from a $\mathsf{C}_{j+1}^{l'_{j+1}}$ that refers to W. $l'_{j+1} = l_{j+1}$ is violated negligibly as ζ_{j+1} is uniform in G_q. $\pi|\tau_j$ originates from a $\mathsf{S}^{l'_s}$. $l'_s = l_s$

is violated negligibly as two server instances compute an identical W with a negligible probability. Thus, Ins does not meet the existential condition with a negligible probability. The uniqueness is simple: two instances in C_j refers to the same W with a negligible probability and two instances in S compute the same W with a negligible probability. □

We now consider $\Pr[\mathbf{Succ}_3(\mathcal{A}, \Gamma_9)|\overline{\mathbf{Succ}_1} \wedge \overline{\mathbf{Succ}_2}]$. Note that none of C_i is corrupted in the **Test** session. Thus, K_{1n} is chosen uniformly random. It follows that the adversary can succeed only with probability 1/2, unless he has seen K_{1n} explicitly. The query to achieve this is only **Reveal**. From Claim 3, only partnered instances will conclude identical session keys. However, neither the tested instance nor its partnered instance is allowed a **Reveal** query. Thus, $\Pr[\mathbf{Succ}_3(\mathcal{A}, \Gamma_9)|\overline{\mathbf{Succ}_1} \wedge \overline{\mathbf{Succ}_2}] = \frac{1}{2} + \mathbf{negl}(\kappa)$.

Claim 3. (Partnering Property) Assume $\mathbf{Succ}_1$ and $\mathbf{Succ}_2$ do not occur. If two instances conclude an identical session key[6] while no involved client is corrupted, then they are partnered except for a negligible probability.

Proof. Assume $\mathsf{C}_i^{l_i}$ and $\mathsf{C}_j^{l_j}$ conclude an identical K_{1n}. As $K_{1,n}$ is uniform in G_q, $\mathsf{C}_i^{l_i}$ and $\mathsf{C}_j^{l_j}$ must refer to an identical record, say W. From the proof of Claim 2, the instance lists induced by $\mathsf{C}_i^{l_i}$ and $\mathsf{C}_j^{l_j}$ is distinct with a negligible probability. Thus, the conclusion follows. □

Thus, $\Pr[\mathbf{Succ}_1(\mathcal{A}, \Gamma)] = \alpha$, then $\Pr[\mathbf{Succ}(\mathcal{A}, \Gamma_9)] = \alpha + \frac{1-\alpha}{2} + \mathbf{negl}(\kappa) \leq \frac{1}{2} + \frac{n_0 Q_{send}}{2|D|} + \mathbf{negl}(\kappa)$. Together with previous Lemmas, we conclude Theorem 1. ■

References

1. M. Abdalla, P. Fouque and D. Pointcheval, Password-Based Authenticated Key Exchange in the Three-Party Setting, *PKC'05*.
2. N. Asokan and P. Ginzboorg, Key Agreement in Ad-hoc Networks, *Computer Communications*, 23(18):1627-1637, 2000.
3. M. Bellare, A. Boldyreva, S. Micali, Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements, *EUROCRYPT 2000*, pp. 259-274.
4. M. Bellare, R. Canetti, and H. Krawczyk, A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols, *STOC 98*, pp. 419-428.
5. M. Bellare, P. Rogaway, Random Oracles are Practical: A Paradigm for Designing Efficient Protocols, *ACM CCS'93*, pp. 62-73.
6. M. Bellare, D. Pointcheval and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attack. *EUROCRYPT'00*, pp. 139-155.
7. S. Bellovin and M. Merritt, Encrypted Key Exchange: Password Based Protocols Secure against Dictionary Attacks, *IEEE Symposium on Research in Security and Privacy 1992*, pp. 72-84.
8. E. Bresson, O. Chevassut and D. Pointcheval, Group Diffie-Hellman Key Exchange Secure against Dictionary Attacks, *Asiacrypt'02*, December 2002.
9. R. Canetti and H. Krawczyk, Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels, *Eurocrypt 2001*, pp. 453-474.
10. R. Canetti, H. Krawczyk, Security Analysis of IKE's Signature-Based Key-Exchange Protocol, *CRYPTO 2002*, pp. 143-161.
11. O. Goldreich, Y. Lindell, Session-Key Generation Using Human Passwords Only, *CRYPTO 2001*, pp. 408-432.
12. L. Gong, M. Lomas, R. Needham and J. Saltzer, Protecting Poorly Chosen Secrets from Guessing Attacks, *IEEE Journal on Selected Areas in Communications*, 11(5): 648-656, 1993.

[6] Here identity should be looked as equality for two random variables.

13. L. Gong, Optimal Authentication Protocols Resistant to Password Guessing Attacks, *Proceedings of the 8th IEEE Computer Security Foundation Workshop*, pages 24-29, 1995.
14. S. Halevi, H. Krawczyk, Public-Key Cryptography and Password Protocols, *ACM Conference on Computer and Communications Security 1998*, pp. 122-131.
15. L. Hua, *Introduction to Number Theory*, Berlin: Springer-verlag, 1982.
16. J. Katz, R. Ostrovsky, M. Yung, Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords, *EUROCRYPT 2001*, pp. 475-494.
17. T. Kwon, M. Kang and J. Song, An Adaptable and Reliable Authentication Protocol for Communication Networks, *Proceedings of IEEE Infocom 1997*, pages 737-744, 1997.
18. D. McGrew and A. Sherman, Key Establishment in Large Dynamic Groups Using One-way Function Trees, *Techinical Report 0755*, TIS Labs at Network Associates, Inc., May 1998.
19. V. Shoup, On Formal Models for Secure Key Exchange, available at http://philby.ucsd.edu/cryptolib/1999.html.
20. Z. Wan, an efficient server-assisted group password-authenticated key exchange protocol, *Manuscript,* 2004.

Appendix 1. Diffie-Hellman Self-reduction.

Now we introduce a self-reduction technique [3, 19]. Let p, q be two large primes with $p = 2q+1$; G_q be the subgroup of order q in $\mathbb{Z}_p^*$. And $g \in G_q \setminus\{1\}$. Thus, $\langle g\rangle = G_q$. Given a bit x and a triple (g^a, g^b, g^c), a self-reduction algorithm Rud_x in [3, 19] can efficiently compute a new triple $(g^{a'}, g^{b'}, g^{c'})$ with the properties in Table 1.

Table 1. Properties of Output from Self-reduction Rud_x

	$x = 0$	$x = 1$
$c = ab$	$a' = a$ & b' random in $\mathbb{Z}_q$ & $c' = a'b'$	a', b' random in $\mathbb{Z}_q$ & $c' = a'b'$
$c \neq ab$	$a' = a$ &b', c' random in $\mathbb{Z}_q$	a', b', c' random in $\mathbb{Z}_q$

For example, if the input is $x = 0$ and a triple (g^a, g^b, g^{ab}), then the output will be $(g^a, g^{b'}, g^{ab'})$, where b' is uniformly random in $\mathbb{Z}_q$. For simplicity, we use $(g^{a'}, g^{b'}, g^{c'}) \leftarrow Rud_x(g^a, g^b, g^c)$ to denote a random output of Rud with input x and (g^a, g^b, g^c).