On the Provable Security of the Dragonfly Protocol

Jean Lancrenon and Marjan Škrobot

Interdisciplinary Centre for Security, Reliability and Trust (SnT),
University of Luxembourg
{jean.lancrenon, marjan.skrobot}@uni.lu

Abstract. **Dragonfly** is a password-authenticated key exchange protocol that was proposed by Harkins [11] in 2008. It is currently a candidate for standardization by the Internet Engineering Task Force, and would greatly benefit from a security proof. In this paper, we prove the security of a very close variant of **Dragonfly** in the random oracle model. It shows in particular that **Dragonfly**'s main flows - a kind of Diffie-Hellman variation with a password-derived base - are sound. We employ the standard Bellare et al. [2] security model, which incorporates forward secrecy.

1 Introduction

Authenticated Key Exchange (AKE) is a cryptographic service run between two or more parties over a network with the aim of agreeing on a secret, high-quality, session key to use in higher-level applications (e.g., to create an efficient and secure channel.) One talks of *Password*-Authenticated Key Exchange (PAKE) if the message flows of the protocol itself are authenticated by means of a *low-entropy password* held by each user. The inherent danger in this setup is its vulnerability to *dictionary attacks*, wherein an adversary - either eavesdropping or impersonating a user - tries to correlate protocol messages with password guesses to determine the correct password being used.

PAKE research is very active. New protocol designs are regularly proposed and analyzed, and PAKE itself has been subject to standardization since at least 2002. In 2008 Harkins proposed **Dragonfly** [11]: Specifically tailored for mesh networks, it is up for IETF (Internet Engineering Task Force) standardization [12]. However, proving the security of **Dragonfly** remains open.

This paper[1] proves secure a protocol similar to the version of **Dragonfly** up for standardization, in the random oracle (RO) model [4]. Thus we can at least assert that the scheme's main flows - a Diffie-Hellman [10] variant with a password-derived base - are sound. **Dragonfly**'s design is similar to that of Jablon's **SPEKE** [13]. MacKenzie having proved **SPEKE** secure in [17], we followed [17]'s proof to structure ours. However, unlike in [17], we incorporated forward secrecy into the analysis, and chose to work in the Bellare et al. model [2]. To our knowledge, this is the first time a protocol employing a password-derived

[1] This paper is the full version of the extended abstract that appears in [16].

Diffie-Hellman base is proven forward-secure and analyzed using [2]. As in [17], **Dragonfly**'s security is based on the Computational Diffie-Hellman (CDH) and Decisional Inverted-Additive Diffie-Hellman (DIDH) assumptions (see Sect. 2.2).

Related Work. PAKE has been heavily studied in the last decade. It began with the works of Bellovin and Merrit [5] and Jablon [13], but with no precise security analysis. Security models in the vein of [3] and [20] were then introduced by Bellare et al. [2] and Boyko et al. [6] respectively, and the number of provably secure schemes - with random oracles (RO) or ideal ciphers [2,7], common reference strings [14,8], universal composability [8], to name a few - has exploded. We refer to Pointcheval's survey [19] for a more complete picture. As for **Dragonfly**, it first appeared in [11]. The attention it has received as an IETF proposal has led it to being broken by Clarke and Hao [9], and subsequently fixed.

Organization. The rest of the paper is structured as follows. In Sect. 2, we recall the commonly-used security model of [2]. Section 3 contains a description of the version of **Dragonfly** we analyze, while the description of the original Dragonfly protocol from [12] can be found in the appendix. Next, Sect. 4 presents the security proof. Finally, the paper is concluded in Sect. 5.

2 Security Model

We use the indistinguishability-based framework of [2], designed for two-party PAKE. In what follows, we will assume some familiarity with the model in [2].

2.1 Model

Participants, Passwords and Initialization. Each principal U that can participate in a PAKE protocol P comes from either the $Client$ or $Server$ set, which are finite, disjoint, nonempty sets whose union is the set ID. We assume that each client $C \in Client$ is in possession of a password π_C, while each server $S \in Server$ holds a vector of the passwords of all clients $\pi_S = \langle \pi_S[C] \rangle_{C \in Client}$. Before the execution of a protocol, an initialization phase occurs, in which public parameters are fixed and a secret π_C, drawn uniformly (and independently) at random from a finite set **Passwords** of size N, is generated for each client and given to all servers.

Protocol Execution. The protocol P is a probabilistic algorithm that defines the way principals behave in response to messages from the environment. In the real world, each principal may run multiple executions of P with different partners, and to model this we allow each principal to have an unlimited number of *instances* executing P in parallel. We denote client instances by C^i and server instances by S^j. Each instance maintains local state (i.e. $state_U^i$, sid_U^i, pid_U^i, sk_U^i, acc_U^i, $term_U^i$) and can be used only once. To assess the security of P, we assume that an adversary $\mathcal{A}$ has complete control of the network. Thus, $\mathcal{A}$ provides the inputs to instances, via the following *queries*:

- **Send**(U^i, M): $\mathcal{A}$ sends message M to instance U^i. As a result, U^i processes M according to P, updates its local state, and outputs a reply. A **Send**$(C^i,$ **Start**$)$ has client C^i output P's first message. This query models active attacks.
- **Execute**(C^i, S^j): This triggers an honest run of P between C^i and S^j, and its transcript is given to $\mathcal{A}$. It covers passive eavesdropping on protocol flows.
- **Reveal**(U^i): $\mathcal{A}$ receives the current value of the session key sk_U^i. $\mathcal{A}$ may do this only if U^i holds a session key. This captures session key leakage.
- **Test**(U^i): A bit b is flipped. If $b = 1$, $\mathcal{A}$ gets sk_U^i. Otherwise, it receives a random string from the session key space. $\mathcal{A}$ may only make one such query at any time during the execution. This query measures sk_U^i's semantic security.
- **Corrupt**(U): π_U is given to $\mathcal{A}$. This models compromise of the long-term key.[2]

Accepting and Terminating. An instance U^i accepts ($acc_U^i = 1$) if it holds a session key sk_U^i, a session ID sid_U^i and a partner ID pid_U^i. An instance U^i terminates ($term_U^i = 1$) if it will not send nor receive any more messages. U^i may accept and terminate *once*.

Partnering. Instances C^i and S^j are partnered if: (1) $acc_C^i = 1$ and $acc_S^j = 1$; (2) $sid_C^i = sid_S^j \neq \perp$; (3) $pid_C^i = S$ and $pid_S^j = C$; (4) $sk_C^i = sk_S^j$; and (5) no other instance accepts with the same sid.

Freshness. Freshness captures the idea that the adversary should not trivially know the session key being tested. An instance U^i is said to be fresh with forward secrecy if: (1) $acc_U^i = 1$; (2) no **Reveal** query was made to U^i nor to its partner U'^j (if it has one); (3) no **Corrupt**(U') query was made before the **Test** query and a **Send**(U^i, M) query was made at some point, where U' is any participant.

Advantage of the Adversary. Now that we have defined freshness and all the queries available to the adversary $\mathcal{A}$, we can formally define the authenticated key exchange (ake) advantage of $\mathcal{A}$ against P. We say that $\mathcal{A}$ wins and breaks the ake security of P, if upon making a **Test** query to a fresh instance U^i that has terminated, $\mathcal{A}$ outputs a bit b', such that $b' = b$ where b is the bit from the **Test** query. We denote the probability of this event by $\mathbf{Succ}_{\mathrm{P}}^{ake}(\mathcal{A})$. The *ake*-advantage of $\mathcal{A}$ in breaking P is

$$\mathbf{Adv}_{\mathrm{P}}^{ake}(\mathcal{A}) = 2\ \mathbf{Succ}_{\mathrm{P}}^{ake}(\mathcal{A}) - 1\ . \tag{1}$$

Authentication. Another of $\mathcal{A}$'s goals is violating authentication. In [2], Bellare et al. define three notions of authentication: client-to-server (**c2s**), server-to-client (**s2c**), and mutual (**ma**). We denote by $\mathbf{Succ}_{\mathrm{P}}^{c2s}(\mathcal{A})$ (respectively, $\mathbf{Succ}_{\mathrm{P}}^{s2c}(\mathcal{A})$) the probability that **c2s** (resp., **s2c**) authentication is violated, which happens if some server (resp., client) has terminated before any **Corrupt** query without

[2] This is the weak-corruption model of [2].

being partnered with a client (resp., server). The adversary is said to violate mutual authentication if there exists some instance that terminates before any **Corrupt** query without a partner. We denote by $\mathbf{Succ}^{ma}_{\mathrm{P}}(\mathcal{A})$ the probability of this event occurring, and the ma-advantage of $\mathcal{A}$ in breaking P is

$$\mathbf{Adv}^{ma}_{\mathrm{P}}(\mathcal{A}) = \mathbf{Succ}^{ma}_{\mathrm{P}}(\mathcal{A}) \ . \tag{2}$$

2.2 Security Assumptions

Here we state the assumptions upon which the security of **Dragonfly** rests. Let $\varepsilon \in [0,1]$, and let $\mathcal{B}$ and $\mathcal{D}$ be probabilistic algorithms running in time t. Let $\mathbb{G}$ be a finite group of prime order q, and g be a generator of $\mathbb{G}$. We say that the assumption holds if there is no (t,ε)-solver for polynomial t (in the security parameter k governing the size of $\mathbb{G}$) and non-negligible ε (also in k).

Computational Diffie-Hellman (CDH). Set $DH_g(g^x, g^y) := g^{xy}$, for any x and y in $\mathbb{Z}_q$. We say that $\mathcal{B}$ is a (t,ε)-CDH solver if

$$\mathbf{Succ}^{cdh}_{g,\mathbb{G}}(\mathcal{B}) := \Pr[\mathcal{B}(g, g^x, g^y) = DH_g(g^x, g^y)] \ \geq \ \varepsilon \ , \tag{3}$$

where x and y are chosen uniformly at random.

Decisional Inverted-Additive Diffie-Hellman (DIDH). For x and y in $\mathbb{Z}^*_q$, where $x + y \neq 0$, set $IDH_g(g^{1/x}, g^{1/y}) := g^{1/(x+y)}$. An algorithm $\mathcal{D}$ is a (t,ε)-DIDH solver if

$$\mathbf{Adv}^{didh}_{g,\mathbb{G}}(\mathcal{D}) := \mathbf{Succ}^{didh}_{g,\mathbb{G}}(\mathcal{D}) - \frac{1}{2} \ \geq \ \varepsilon \ , \tag{4}$$

where $\mathbf{Succ}^{didh}_{g,\mathbb{G}}(\mathcal{D}) := \Pr[b' = b]$ in the following game. First, x, y, and z are chosen uniformly at random and a bit b is flipped. Let $X := g^{1/x}$ and $Y := g^{1/y}$. If $b = 0$, set $Z := g^{1/z}$, and if $b = 1$, set $Z := IDH_g(X, Y)$. $\mathcal{D}$ gets as input (g, X, Y, Z), and outputs bit b'.

The DIDH assumption is less-known than the CDH one. It states that it is hard to tell apart $g^{1/(x+y)}$ and a random $g^{1/z}$ when given $g^{1/x}$ and $g^{1/y}$. [17] shows that DIDH is as hard as the Decisional Diffie-Hellman problem in generic groups. For a nice overview of the relations between the DIDH assumption and other discrete-logarithm-style assumptions we refer the reader to [1].

3 The Dragonfly Protocol

We first fix some notation and then describe the version of **Dragonfly** to analyze. Its cryptographic core is a Diffie-Hellman key exchange similar to the one used in **SPEKE** [13], where a function of the password is the base for group values.

Notation. Let $\mathbb{G}$ be a finite multiplicative group of prime order q, and k be the security parameter. When we sample elements from $\mathbb{Z}_q$, it is understood that they are viewed as integers in $[1 \dots q]$, and all operations on these are performed mod q. Let H_0 be a full-domain hash mapping $\{0,1\}^*$ to $\mathbb{G}$. We also define a hash function H_1 from $\{0,1\}^*$ to $\{0,1\}^{3k}$. $a \leftarrow A$ denotes selecting a uniformly at random from the set A. Let the function $\mathrm{Good}(E, s)$ be true iff: (1) $s \in [1 \dots q]$ and (2) $E \in \mathbb{G}$. We assume the existence of an efficient algorithm to perform the latter check; this is important, as it prevents instantiation-specific attacks, like the small subgroup attack in [9].

Protocol Description. A high-level protocol description is shown in Fig. 1. At initialization, the password is chosen at random from **Passwords** and given to the client and server. Then, both parties compute a base $PW = H_0(C, S, \pi)$ for Diffie-Hellman values, where C and S are ID strings.

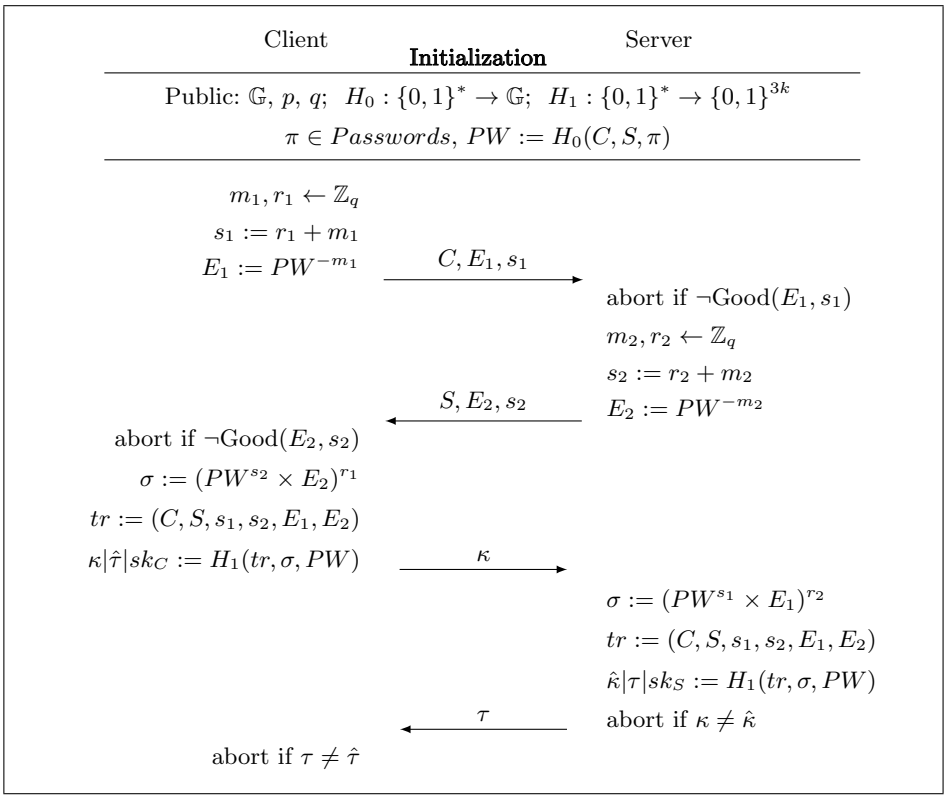

Fig. 1: The Dragonfly protocol.

In nutshell, the protocol runs in two phases. In the first phase, each participant chooses a random exponent r_i and mask m_i, computes their sum $s_i \in \mathbb{Z}_q$ and the

group element $E_i := PW^{-m_i}$, and sends the commit message (ID, E_i, s_i), where $i = 1, 2$. Upon receiving this message, Good(E_i, s_i) is called to check its validity. At this point, the session IDs sid_C and sid_S are set to $(C, S, s_1, s_2, E_1, E_2)$ for each participant. In the second phase, both participants derive the Diffie-Hellman value $\sigma = PW^{r_1 r_2}$. This is followed by a computation of a hash value (using the derived σ value), parsed into three k-bit strings: an authenticator for each participant and the session key sk. Then, the authenticators are exchanged. If the received authenticator is valid, the participant accepts and terminates the execution, saving session key sk. Otherwise, it aborts, deleting its state.

Remarks. We point out here the main areas where the presented protocol slightly differs from the IETF proposal (see Appendix A). First of all, we do not model the "hunting-and-pecking" procedure explicitly, but this is not a problem here. As pointed out in the proposal, "hunting-and-pecking" is just one way among others to deterministically obtain a base group element from a password. Thus, simply using a random oracle taking as input the participants' identities in addition to the password is appropriate.

The procedure we use to compute the confirmation codes and the session keys is not that of the proposal. In particular, our construction makes all of these direct functions of the shared secret, both identities, the main protocol's message flows, and the password element. This is similar to the PAK and PPK protocols [6], for instance, as well as in MacKenzie's analysis of **SPEKE** [17]. In our view, it is more prudent to follow this pattern, as either removing identities - or replacing them with generic "role" strings - can lead to attacks, e.g. [18] and [1]. Thus, we recommend adding the receiver's identity in the IEFT proposal's computation of the "confirm" message.

Finally, the protocol could have been dropped to three flows, but we chose to keep it four, for two reasons. First, this way we stay close to the IETF protocol, and secondly, despite reducing communication efficiency, four-flow PAKEs - in which the first two flows commit to a shared password and the second two are proofs-of-possession of the session key - are by design secure against many-to-many guessing attacks on the server side, see [15].

4 Security proof of Dragonfly protocol

We now present a proof of security for **Dragonfly** in the RO model [4]. We show that **Dragonfly** distributes semantically secure session keys, provides mutual authentication, and enjoys forward secrecy. We also adopt the convenient notations of [7].

Theorem 1. *We consider **Dragonfly** as described in Sect. 3, with a password set of size N. Let $\mathcal{A}$ be an adversary that runs in time at most t, and makes at most n_{se} **Send** queries, n_{ex} **Execute** queries, and n_{h0} and n_{h1} RO queries to H_0 and H_1, respectively. Then there exist two algorithms $\mathcal{B}$ and $\mathcal{D}$ running in time t' such that $\mathbf{Adv}^{ake}_{\texttt{dragonfly}}(\mathcal{A}) \leq T$ and $\mathbf{Adv}^{ma}_{\texttt{dragonfly}}(\mathcal{A}) \leq T$ where*

$$T := \frac{6n_{se}}{N} + \frac{4(n_{se}+n_{ex})(2n_{se}+n_{ex}+n_{h1})}{q^2} + \frac{n_{h0}^2+2n_{h1}}{q} + \frac{n_{h1}^2+2n_{se}}{2^k} + 2n_{h1}(1+{n_{se}}^2) \times Succ^{cdh}_{PW,\mathbb{G}}(\mathcal{B}) + 4n_{h0}^3 \times \left(\boldsymbol{Adv}^{didh}_{g,\mathbb{G}}(\mathcal{D}) + \frac{n_{h1}^3+3n_{se}}{q} \right) \quad (5)$$

and where $t' = O(t + (n_{se} + n_{ex} + n_{ro})t_{exp})$ with t_{exp} being a time required for exponentiation in $\mathbb{G}$.

Proof. Our proof is given as a sequence of games $\mathbf{G}_0, ..., \mathbf{G}_4$. Our goal is to prove that **Dragonfly** resists offline dictionary attacks, i.e. that $\mathcal{A}$'s advantage is proportional to that of the easily detected "dummy" online guesser. We define events, corresponding to $\mathcal{A}$ attacking the protocol in game $\mathbf{G}_m$ and breaking semantic security, and **c2s** and **s2c** authentication, for $m = 0, ..., 4$.

- $\mathbf{S}_m$ occurs if $\mathcal{A}$ returns b' equal to the bit b chosen in the **Test** query.
- $\mathbf{Auth}^{c2s}_m$ occurs if an S^j terminates saving sk_S as a state without being partnered with some C^i.
- $\mathbf{Auth}^{s2c}_m$ occurs if a C^i terminates saving sk_C as a state without being partnered with some S^j.

Throughout the proof, we call $\mathcal{A}$'s oracle query of the form $H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$ **bad** if $\sigma = DH_{PW}(m, \mu)$ (where $m = PW^{s_1}E_1$ and $\mu = PW^{s_2}E_2$), and $\text{Good}(E_1, s_1)$ and $\text{Good}(E_2, s_2)$ are true. Also, we denote by π_C the value of the password selected for C and by $PW_{C,S}$ the value of the base derived from it with server S. The number of instances that $\mathcal{A}$ can activate and of hash queries that $\mathcal{A}$ can make are bounded by t. In addition, in case $\mathcal{A}$ does not output b' after time t, b' is chosen randomly. Let us now proceed with a detailed proof.

Game $\mathbf{G}_0$: This game is our starting point, with **Dragonfly** defined as in Fig. 1. $\mathcal{A}$ may make **Send**, **Execute**, **Reveal**, **Corrupt**, and **Test** queries and these queries are simulated as shown in Fig(s). 2, 3, and 4. From Def. 1 we have

$$\mathbf{Adv}^{ake}_{\texttt{dragonfly}}(\mathcal{A}) = 2\ \Pr[\mathbf{S}_0] - 1\ . \quad (6)$$

Game $\mathbf{G}_1$: This is our first simulation, in which hash queries[3] to H_0, H_1 and H_1' are answered by maintaining lists $\mathcal{L}_{h0}, \mathcal{L}_{h1}$ and $\mathcal{L}'_{h1}$, respectively (see Fig. 5). The simulator also maintains a separate list $\mathcal{L}_{\mathcal{A}}$ of all hash queries asked by $\mathcal{A}$. Note that we assume that the simulator knows the discrete logarithms of the outputs of H_0 queries. The simulator also keeps track of all honestly exchanged protocol messages in the list $\mathcal{L}_{\mathcal{P}}$. We say that a client instance C^i and a server instance S^j are paired if $((C, E_1, s_1), (S, E_2, s_2)) \in \mathcal{L}_P$. We can easily see that this simulation is perfectly indistinguishable from the attack in $\mathbf{G}_0$. Thus,

$$\Pr[\mathbf{S}_1] = \Pr[\mathbf{S}_0]\ . \quad (7)$$

Send queries made to a client instance C^i are answered as follows:

- A **Send**(C^i, **Start**) query is executed according to the following rule:
  - ⋆ **Rule C1**$^{(1)}$
    Choose an ephemeral exponent $r_1 \leftarrow \mathbb{Z}_q$ and a mask $m_1 \leftarrow \mathbb{Z}_q$, compute $s_1 := r_1 + m_1$ and $E_1 := PW^{-m_1}$.

  The client instance C^i then replies to the adversary $\mathcal{A}$ with (C, E_1, s_1) and goes to an expecting state EC_1.
- If the instance C^i is in the expecting state EC_1, a received **Send**(C^i, (S, E_2, s_2)) query is first parsed and Good(E_2, s_2) is called. If the check passes, the instance continues processing the query according to the following rules:
  - ⋆ **Rule C2**$^{(1)}$
    Compute $\sigma := (PW^{s_2} \times E_2)^{r_1}$.
  - ⋆ **Rule C3**$^{(1)}$
    Compute $\kappa|\hat{\tau}|sk_C := H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$.

  The instance C^i accepts, replies to $\mathcal{A}$ with κ, and goes to an expecting state EC_2. Otherwise, it terminates (rejecting), saving no state.
- In case C^i is in the expecting state EC_2, a **Send**(C^i, τ) query is processed according to the following rule:
  - ⋆ **Rule C4**$^{(1)}$
    Check if $\tau = \hat{\tau}$. If so, the instance terminates, saving sk_C as a state.

  If the equality does not hold, the instance terminates (rejecting), saving no state.

Fig. 2: Simulation of the Send queries to the client.

Game $\mathbf{G_2}$: In this game, collisions on the outputs of H_0 queries and collisions on the partial transcripts $((C, E_1, s_1), (S, E_2, s_2))$ are avoided. Let list $\mathcal{L}_\mathcal{R}$ keep track of the replies generated by client and server instances as answers to **Send** queries. We abort if a pair (E_1, s_1) generated by a client instance is already in the list $\mathcal{L}_\mathcal{R}$ as a result of previous **Send** or **Execute** queries, or in the list $\mathcal{L}_\mathcal{A}$ as an input to an H_1 query. Similarly, we abort in case a pair (E_2, s_2) generated by a server instance is already in $\mathcal{L}_\mathcal{R}$ or $\mathcal{L}_\mathcal{A}$.

⋆ **Rule C1**$^{(2)}$
Choose an ephemeral exponent $r_1 \leftarrow \mathbb{Z}_q$ and a mask $m_1 \leftarrow \mathbb{Z}_q$, compute $s_1 := r_1 + m_1$ and $E_1 := PW^{-m_1}$. If $(E_1, s_1) \in \mathcal{L}_\mathcal{R} \cup \mathcal{L}_\mathcal{A}$, abort the game.

⋆ **Rule S1**$^{(2)}$
Choose an ephemeral exponent $r_2 \leftarrow \mathbb{Z}_q$ and a mask $m_2 \leftarrow \mathbb{Z}_q$, compute $s_2 := r_2 + m_2$ and $E_2 := PW^{-m_2}$. If $(E_2, s_2) \in \mathcal{L}_\mathcal{R} \cup \mathcal{L}_\mathcal{A}$, abort the game.

Additionally, we abort in case of collisions on H_0 outputs. This event's probability is bounded by the birthday paradox $n_{h0}^2/2q$.

[3] The private oracle $H_1' : \{0,1\}^* \rightarrow \{0,1\}^{3k}$ will be used in the later, starting from the $\mathbf{G_3}$.

Send queries made to a server instance S^j are answered as follows:

- A **Send**$(S^j, (C, E_1, s_1))$ query is first parsed and then Good(E_1, s_1) is called. If both values are valid, the instance continues processing the query according to the following rules:
  - ⋆ **Rule S1**$^{(1)}$
    Choose an ephemeral exponent $r_2 \leftarrow \mathbb{Z}_q$ and a mask $m_2 \leftarrow \mathbb{Z}_q$, compute $s_2 := r_2 + m_2$ and $E_2 := PW^{-m_2}$.

  The server instance S^j then replies to the adversary $\mathcal{A}$ with (S, E_2, s_2) and goes to an expecting state ES_1. Otherwise, it terminates (rejecting), saving no state.
- If the instance S^j is in the expecting state ES_1, a **Send**(S^j, κ) query is executed according to the following rules:
  - ⋆ **Rule S2**$^{(1)}$
    Compute $\sigma := (PW^{s_1} \times E_1)^{r_2}$.
  - ⋆ **Rule S3**$^{(1)}$
    $\hat{\kappa}|\tau|sk_S := H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$.
  - ⋆ **Rule S4**$^{(1)}$
    Check if $\kappa = \hat{\kappa}$. If so, S^j accepts, replies with τ, and terminates while saving sk_S as a state.

  If the equality does not hold, the S^j terminates (rejecting), saving no state.

Fig. 3: Simulation of the Send queries to the server.

⋆ **Rule** $H_0^{(2)}$
Choose $\alpha \leftarrow \mathbb{Z}_q$. Compute $r := g^\alpha$ and write the record (w, r, α) to $\mathcal{L}_{h0}$. If $(*, r, *) \in \mathcal{L}_{\mathcal{A}}$, abort the game.

The rule modifications in this game ensure the uniqueness of honest instances and that distinct passwords do not map to the same base PW. So we have:

$$|\Pr[S_2] - \Pr[S_1]| \leq \frac{2(n_{se} + n_{ex})(2n_{se} + n_{ex} + n_{h1})}{q^2} + \frac{n_{h0}^2}{2q} \ . \tag{8}$$

Game G$_3$: In this game, we first define event **Corrupted** that occurs if the adversary makes a **Corrupt** query while the targeted client and server instance are not paired. From now on, if **Corrupted** is false, instead of using H_1 to compute session keys and authenticators, the simulator uses a private oracle H_1'. The rules change as follows:

⋆ **Rule C3**$^{(3)}$
If **Corrupted** is false, compute $\kappa|\hat{\tau}|sk_C := H_1'(C, S, s_1, s_2, E_1, E_2)$. Otherwise, compute $\kappa|\hat{\tau}|sk_C := H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$.

⋆ **Rule S3**$^{(3)}$
If **Corrupted** is false , compute $\hat{\kappa}|\tau|sk_S := H_1'(C, S, s_1, s_2, E_1, E_2)$. Otherwise, compute $\hat{\kappa}|\tau|sk_C := H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW)$.

An **Execute**(C^i, S^j) query is simulated by successively running the honest simulations of **Send** queries. After the completion, the transcript is given to the adversary.

As a result of the **Reveal**(U^i) query, the simulator returns the session key (either sk_C or sk_S) to $\mathcal{A}$, only in case the instance U^i has already computed the key and accepted.

As a result of the **Corrupt**(U) query, if $U \in Client$ the simulator returns the password π_C, and otherwise the vector of passwords $\pi_S = \langle \pi_S[C] \rangle_{C \in Client}$.

As a result of the **Test**(U^i) query, the simulator flips a bit b. If $b = 1$, it returns session key sk_U^i to $\mathcal{A}$. Otherwise, $\mathcal{A}$ receives a random string drawn from $\{0,1\}^k$.

Fig. 4: Simulation of the Execute, Reveal, Corrupt, and Test queries.

$\boldsymbol{H_0}$: For each hash query $H_0(w)$, if the same query was previously asked, the simulator retrieves the record (w, r, α) from the list $\mathcal{L}_{h0}$ and answers with r. Otherwise, the answer r is chosen according to the following rule:

⋆ **Rule** $H_0^{(1)}$
Choose $\alpha \leftarrow \mathbb{Z}_q$. Compute $r := g^\alpha$ and write the record (w, r, α) to $\mathcal{L}_{h0}$.

$\boldsymbol{H_1}$: For each hash query $H_1(w)$ (resp. $H_1'(w)$), if the same query was previously asked, the simulator retrieves the record (w, r) from the list $\mathcal{L}_{h1}$ (resp. $\mathcal{L}_{h1}'$) and answers with r. Otherwise, the answer r is chosen according to the following rule:

⋆ **Rule** $H_1^{(1)}$
Choose $r \leftarrow \{0,1\}^{3k}$, write the record (w, r) in the list $\mathcal{L}_{h1}$ (resp. $\mathcal{L}_{h1}'$), and answer with r.

Fig. 5: Simulation of the hash functions.

Then, since the shared secret σ and base PW are no longer used in above computations in case the event **Corrupted** is false, we can further modify the following rules:

⋆ **Rule C2**$^{(3)}$
If **Corrupted** is false, do nothing. Otherwise, compute $\sigma := (PW^{s_2} \times E_2)^{r_1}$.

⋆ **Rule S2**$^{(3)}$
If **Corrupted** is false, do nothing. Otherwise, compute $\sigma := (PW^{s_1} \times E_1)^{r_2}$.

⋆ **Rule C1**$^{(3)}$
Choose $\psi_1, s_1 \leftarrow \mathbb{Z}_q$ and compute $E_1 := g^{\psi_1}$. If $(E_1,\, s_1) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort.

⋆ **Rule S1**$^{(3)}$
Choose $\psi_2, s_2 \leftarrow \mathbb{Z}_q$ and compute $E_2 := g^{\psi_2}$. If $(E_2,\, s_2) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort.

Note that after the above modification the simulator can determine correct and incorrect password guesses and answer perfectly to all queries using the ψ_1, ψ_2, and α values and the lists $\mathcal{L}_{h0}$, $\mathcal{L}_{h1}$, $\mathcal{L}_{h1}'$, $\mathcal{L}_{\mathcal{A}}$, $\mathcal{L}_{\mathcal{P}}$, and $\mathcal{L}_{\mathcal{R}}$. Also, the values s_1, s_2, E_1, E_2 obtained after applying rules $C1^{(3)}$ and $S1^{(3)}$ are identically distributed to those generated in game $\mathbf{G_2}$.

Now that the password-derived base is absent from protocol executions (if **Corrupted** is false[4]), we can dismiss the event that $\mathcal{A}$ has been lucky in guessing the right $PW_{C,S}$ without making the corresponding H_0 query. Hence we abort the simulation if the adversary $\mathcal{A}$ submits a $H_1(C,\ S,\ *,\ *,\ *,\ *,\ *,\ PW_{C,S})$ query without prior $H_0(C, S, \pi_C)$ query. The probability of this event occurring is n_{h1}/q.

⋆ **Rule** $H_1^{(3)}$
If $w = (C, S, *, *, *, *, *, PW_{C,S})$, $((C, S, \pi_C), PW_{C,S}) \notin \mathcal{L}_{\mathcal{A}}$, and **Corrupted** is false, abort. Otherwise, choose $r \leftarrow \{0,1\}^{3k}$, write the record (w, r) in the list $\mathcal{L}_{h1}$, and answer with r.

Next, we avoid the cases where the adversary $\mathcal{A}$ may have guessed one of the authenticators (κ or τ) without having made an appropriate H_1 query (when **Corrupted** is false). A "lucky guess" occurs if $\mathcal{A}$ submits a **Send**(S^j, κ) query with the correct authenticator κ to an unpartnered server instance S^j without previously submitting a **bad** $H_1(C,\ S,\ s_1,\ s_2,\ E_1,\ E_2,\ \sigma,\ PW_{C,S})$ query. In this case S^j aborts, even though it should have accepted. Similarly, if $\mathcal{A}$ submits a **Send**(C^i, τ) query with the correct τ to an unpartnered C^i without having submitted a **bad** $H_1(C,\ S,\ s_1,\ s_2,\ E_1,\ E_2,\ \sigma,\ PW_{C,S})$ query, C^i aborts.

⋆ **Rule S4**$^{(3)}$
Check if $\kappa = \hat{\kappa}$. If so, check if $((C, S, s_1, s_2, E_1, E_2, \sigma, PW_{C,S}), \kappa) \notin \mathcal{L}_{\mathcal{A}}$, where $\mathrm{Good}(E_1, s_1)$ is true, $\sigma = DH_{PW_{C,S}}(m, \mu)$, and **Corrupted** is false. If the latter check is true, the server instance S^j aborts. Otherwise, S^j accepts, replies to the adversary with τ, and terminates while saving sk_S as a state.

⋆ **Rule C4**$^{(3)}$
Check if $\tau = \hat{\tau}$. If so, check if $((C, S, s_1, s_2, E_1, E_2, \sigma, PW_{C,S}), \tau) \notin \mathcal{L}_{\mathcal{A}}$, where $\mathrm{Good}(E_2, s_2)$ is true, $\sigma = DH_{PW_{C,S}}(m, \mu)$, and **Corrupted** is false. If the latter check is true, the client instance C^i aborts. Otherwise, C^i terminates, saving sk_C as a state.

Since the authenticators are computed using a private random oracle H_1' (when **Corrupted** is false), we can argue that the adversary can not do better than a random guess per an authentication attempt via **Send** query. Therefore, the probability of "lucky guessing" is bounded by $n_{se}/2^k$.

Without the collisions on the partial transcripts and the "lucky guesses" on the password-derived base and authenticators, one can see that $\mathcal{A}$ has to make the specific combination of H_0 and H_1 hash queries for games $\mathbf{G_2}$ and $\mathbf{G_3}$ to be distinguished. Let **AskH1**$_3$ be the event that $\mathcal{A}$ makes the **bad** query $H_1(C,\ S,\ s_1,\ s_2,\ E_1,\ E_2,\ \sigma,\ PW_{C,S})$ for some transcript $((C, E_1, s_1), (S, E_2, s_2), \kappa, \tau)$, where $H_0(C, S, \pi_C)$ has been already made. Depending on how the transcript is generated, we distinguish between four disjoint sub-cases **AskH1**$_3$:

[4] Notice that in case **Corrupted** is true, a password-derived base is still used in H_1 computations, hence we can not apply the same argument.

– **AskH1-Passive**$_3$: $((C, E_1, s_1), (S, E_2, s_2), \kappa, \tau)$ comes from an honest execution between C^i and S^j (via an **Execute**(C^i, S^j) query);
– **AskH1-Paired**$_3$: $((C, E_1, s_1), (S, E_2, s_2))$ comes from an honest execution between C^i and S^j, while (κ, τ) may come from $\mathcal{A}$;
– **AskH1-withC**$_3$: before any **Corrupt** query, $\mathcal{A}$ interacts with C^i, so (C, E_1, s_1) is generated by C^i, while (S, E_2, s_2) is not from S^j;
– **AskH1-withS**$_3$: before any **Corrupt** query, $\mathcal{A}$ interacts with S^j, so (S, E_2, s_2) is generated by S^j, while (C, E_1, s_1) is not from C^i.

Since session key(s) are computed using the private oracle H_1', the only way $\mathcal{A}$ can break semantic security is via a **Reveal** query to honest instances that generated the same transcript $((C, E_1, s_1), (S, E_2, s_2), \kappa, \tau)$, a case we dismissed in $\mathbf{G_2}$. Thus,

$$\Pr[\mathbf{S}_3] = \frac{1}{2} \ , \quad |\Pr[\mathbf{S}_3] - \Pr[\mathbf{S}_2]| \leq \frac{n_{h1}}{q} + \frac{n_{se}}{2^k} + \Pr[\mathbf{AskH1}_3] \ . \tag{9}$$

Similarly - and as already previously mentioned - the authenticators are computed using H_1' as well, and due to $\mathbf{G_2}$, $\mathcal{A}$ cannot reuse authenticators from other instances. Thus,

$$\Pr[\mathbf{Auth}_3^{c2s}] \leq \frac{n_{se}}{2^k} \ , \quad \Pr[\mathbf{Auth}_3^{s2c}] \leq \frac{n_{se}}{2^k} \ . \tag{10}$$

Game $\mathbf{G_4}$: In this game, we estimate the probability of the event **AskH1**$_3$ occurring and thus conclude the proof. Notice that the probability of **AskH1** occurring does not change between games $\mathbf{G_3}$ and $\mathbf{G_4}$. We also have that

$$\Pr[\mathbf{S}_4] = \Pr[\mathbf{S}_3] \ , \quad \Pr[\mathbf{Auth}_4^{c2s\ (s2c)}] = \Pr[\mathbf{Auth}_3^{c2s\ (resp.,\ s2c)}] \ . \tag{11}$$

Since all the sub-cases of **AskH1**$_4$ are disjoint, we will treat them independently:

The following lemma upper bounds the probability of **AskH1-Passive**$_4$:

Lemma 1. *For any $\mathcal{A}$ running in time t that asks a bad query $H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW_{C,S})$ for some transcript $((C, E_1, s_1), (S, E_2, s_2), \kappa, \tau)$ that comes from an honest execution between C^i and S^j, there is an algorithm $\mathcal{B}$ running in time $t' = O(t + (n_{se} + n_{ex} + n_{ro})t_{exp})$ that can solve the CDH problem:*

$$\Pr[\mathbf{AskH1\text{-}Passive}_4] \leq n_{h1} \times Succ^{cdh}_{PW,\mathbb{G}}(\mathcal{B}) \ . \tag{12}$$

Proof. We construct an algorithm $\mathcal{B}$ that, for given random Diffie-Hellman values $\langle X, Y \rangle$ such that $X \leftarrow g^x$ and $Y \leftarrow g^y$, attempts to break the CDH assumption (i.e. computes Z such that $Z = DH_g(X, Y)$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{B}$ simulates the protocol for $\mathcal{A}$ with the modification of the rules **C1** and **S1** in case an **Execute**(C^i, S^j) query was made:

⋆ **Rule C1**$^{(4)}_{exe}$
Choose $\psi_1, s_1 \leftarrow \mathbb{Z}_q$ and compute $E_1 := Xg^{\psi_1}$. If $(E_1, s_1) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

⋆ **Rule S1**$^{(4)}_{exe}$

Choose $\psi_2, s_2 \leftarrow \mathbb{Z}_q$ and compute $E_2 := Yg^{\psi_2}$. If $(E_2,\, s_2) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

After the game ends, for every $H_1(C,\, S,\, s_1,\, s_2,\, E_1, E_2,\, \sigma,\, PW_{C,S})$ query the adversary $\mathcal{A}$ makes, where the values s_1, s_2, E_1 and E_2 were generated by honest client and server instances (after an **Execute**$(C^i,\, S^j)$ query), the password-derived base is correct, and the corresponding $H_0(C, S, \pi_C)$ query was made,

$$(\sigma Y^{-\frac{\psi_1}{\alpha}} X^{-\frac{\psi_2}{\alpha}} {E_2}^{-s_1} {E_1}^{-s_2} g^{-\frac{\psi_1\psi_2}{\alpha}} g^{-s_1 s_2 \alpha})^{\alpha} \tag{13}$$

is added to the list $\mathcal{L}_Z$ of possible values for $Z = DH_g(X, Y)$. Equation 13 follows from the fact that a base $PW := g^{\alpha}$ is generated in such a way that the discrete logarithm α is known. Thus, the Diffie-Hellman values X and Y can be represented as $PW^{\frac{x}{\alpha}}$ and $PW^{\frac{y}{\alpha}}$, respectively. So we have:

$$\begin{aligned} \sigma &= DH_{PW}(E_2 PW^{s_2}, E_1 PW^{s_1}) \\ &= DH_{PW}(PW^{\frac{y+\psi_2}{\alpha}} PW^{s_2}, PW^{\frac{x+\psi_1}{\alpha}} PW^{s_1}) \\ &= Z^{\frac{1}{\alpha}} Y^{\frac{\psi_1}{\alpha}} X^{\frac{\psi_2}{\alpha}} {E_2}^{s_1} {E_1}^{s_2} g^{\frac{\psi_1\psi_2}{\alpha}} g^{s_1 s_2 \alpha} \ . \end{aligned} \tag{14}$$

From the adversary's view, the simulation $\mathcal{B}$ runs is indistinguishable from the protocol in the game $\mathbf{G_3}$ up to the point **AskH1-Passive**$_4$ occurs, and in that case, the correct $DH_g(X, Y)$ value is added to the list $\mathcal{L}_Z$ of size at most n_{h1}. The running time of $\mathcal{B}$ is $t' = O(t + (n_{ex} + n_{ro} + n_{se})t_{exp})$. Thus, Lemma 1 follows from the fact that the probability of $\mathcal{B}$ breaking CDH assumption is at least $\Pr[\textbf{AskH1-Passive}_4]/n_{h1}$. □

The next lemma bounds the chance of **AskH1-Paired**$_4$ occurring:

Lemma 2. *For any $\mathcal{A}$ running in time t that asks a bad query $H_1(C,\, S,\, s_1,\, s_2,\, E_1, E_2,\, \sigma,\, PW_{C,S})$ for some partial transcript $((C, E_1, s_1), (S, E_2, s_2)) \in \mathcal{L}_{\mathcal{P}}$ that comes from an honest execution between C^i and S^j, there is an algorithm $\mathcal{B}$ running in time $t' = O(t + (n_{se} + n_{ex} + n_{ro})t_{exp})$ that can solve the CDH problem:*

$$\Pr[\textbf{AskH1-Paired}_4] \leq n_{se}^2 n_{h1} \times Succ^{cdh}_{PW,\mathbb{G}}(\mathcal{B}) \ . \tag{15}$$

Proof. The proof is similar to the previous one, except that the simulator needs to guess the client and server instances whose execution is going to be tested. The reason for this comes from the fact that the private exponents of all the instances would be unknown to the simulator if we applied the same reduction as in the proof of Lemma 1. The problem in the simulation could arise in case the adversary sends the authenticator after making the **Corrupt** query. Therefore, if the simulator makes the right guess, the given random Diffie-Hellman values will be inserted in the instances that are fresh (no **Corrupt** query). The reduction goes as follows.

We construct an algorithm $\mathcal{B}$ that, for given random Diffie-Hellman values $\langle X, Y\rangle$ such that $X \leftarrow g^x$ and $Y \leftarrow g^y$, attempts to solve the CDH assumption (i.e. computes Z such that $Z = DH_g(X, Y)$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{B}$ chooses distinct random indexes $b_1, b_2 \leftarrow \{1, 2, \ldots, n_{se}\}$ and simulates the protocol for $\mathcal{A}$ with the modification of the rule $\mathbf{C1}^{(3)}$ in case of a b_1th **Send**$(C^i,$ **Start**$)$ query and the rule $\mathbf{S1}^{(3)}$ in case of a b_2th **Send**$(S^j, (C, E_1, s_1))$ query:

⋆ **Rule C1**$^{(4)}$

For the b_1th query choose $s_1 \leftarrow \mathbb{Z}_q$ and set $E_1 := X$. Otherwise, choose $\psi_1, s_1 \leftarrow \mathbb{Z}_q$ and compute $E_1 := g^{\psi_1}$. In any case, if $(E_1,\ s_1) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

⋆ **Rule S1**$^{(4)}$

For the b_2th query choose $s_2 \leftarrow \mathbb{Z}_q$ and set $E_2 := Y$. Otherwise, choose $\psi_2, s_2 \leftarrow \mathbb{Z}_q$ and compute $E_2 := g^{\psi_2}$. In any case, if $(E_2,\ s_2) \in \mathcal{L}_{\mathcal{R}} \cup \mathcal{L}_{\mathcal{A}}$, abort the game.

After the game ends, for every $H_1(C,\ S,\ s_1,\ s_2,\ E_1, E_2,\ \sigma,\ PW_{C,S})$ query the adversary $\mathcal{A}$ makes, where pairs $(E_1,\ s_1)$ and $(E_2,\ s_2)$ were generated after b_1th and b_2th **Send** query, the password-derived base is correct, and the corresponding $H_0(C, S, \pi_C)$ query was made,

$$(\sigma Y^{-s_1} X^{-s_2} g^{-s_1 s_2 \alpha})^{\alpha} \tag{16}$$

is added to the list $\mathcal{L}_Z$ of possible values for $DH_g(X, Y)$ of size at most n_{h1}.

From the adversary's view, the simulation $\mathcal{B}$ runs is indistinguishable until the adversary triggers **AskH1-Paired**$_4$. The probability that $\mathcal{B}$ will guess the correct client instance, the correct server instance, and the correct Z value from $\mathcal{L}_Z$ is at least $1/(n_{se}^2 n_{h1})$. The running time of $\mathcal{B}$ is $t' = O(t + (n_{ex} + n_{ro} + n_{se})t_{exp})$. Thus, the Lemma 2 follows from the fact that the probability of $\mathcal{B}$ solving the CDH assumption is at least $\Pr[\textbf{AskH1-Paired}_4]/(n_{se}^2 n_{h1})$. □

Before estimating the probability of **AskH1-withS**$_4$ occurring, we evaluate that of **Coll**$_S$, which happens if $\mathcal{A}$ makes two explicit password guesses at the same server instance. Since there are no collisions on H_0 outputs, the only way for $\mathcal{A}$ to accomplish this is if a collision occurs on the first k-bits of two H_1 queries made by $\mathcal{A}$, with $PW_1 \neq PW_2$. The probability of this occurring is bounded by the birthday paradox $n_{h1}^2/2^{k+1}$.

⋆ **Rule** $H_1^{(4)}$

If $w = (C, S, *, *, *, *, *, PW_{C,S})$, $((C, S, \pi_C), PW_{C,S}) \notin \mathcal{L}_{\mathcal{A}}$, and **Corrupted** is false or if **Coll**$_S$ event occurs abort. Otherwise, choose $r \leftarrow \{0,1\}^{3k}$, write the record (w, r) in the list $\mathcal{L}_{h1}$, and answer with r.

Now, without any collision on H_0 and H_1 oracles, each authenticator κ coming from $\mathcal{A}$ via a **Send** query corresponds only to one password π. Therefore,

$$\Pr[\textbf{AskH1-withS}_4] \leq \frac{n_{se}}{N} \ . \tag{17}$$

To bound the probability of **AskH1-withC**$_4$, we first bound the probability of **Coll**$_C$, which happens if $\mathcal{A}$ makes three implicit password guesses against the same client instance. The following lemma gives such a bound:

Lemma 3. *For any $\mathcal{A}$ running in time t that asks at least three bad H_1 queries with distinct values of PW for the same transcript ((C, E_1, s_1), (S, E_2, s_2), κ, τ), generated in a communication between $\mathcal{A}$ and C^i, there exists an algorithm $\mathcal{D}$ running in time $t' = O(t + (n_{se} + n_{ex} + n_{ro})t_{exp})$ that can solve the DIDH problem:*

$$\Pr[\mathbf{Coll}_C] \leq 2n_{h0}^3 \times \left(\mathbf{Adv}_{g,\mathbb{G}}^{didh}(\mathcal{D}) + \frac{n_{h1}^3 + 3n_{se}}{2q} \right) \quad . \tag{18}$$

Proof. This lemma actually shows that the DIDH assumption prevents the adversary from making more than *two* password guesses per online attempt on the client. We reduce from DIDH as follows.

We construct an algorithm $\mathcal{D}$ that given a triple $\langle X, Y, Z \rangle$ as input, where $X \leftarrow g^{1/x}$, $Y \leftarrow g^{1/y}$ and $Z \in \mathbb{G}$, attempts to break the DIDH assumption (i.e. determine whether Z is random or $Z = IDH_g(X, Y) = g^{1/(x+y)}$) by running the adversary $\mathcal{A}$ as a subroutine. The algorithm $\mathcal{D}$ chooses pair-wise distinct random indexes d_1, d_2, $d_3 \leftarrow \{1, 2, \ldots, n_{h0}\}$, chooses random non-zero exponents u_1, u_2, u_3 in $\mathbb{Z}_q$, and simulates the protocol for $\mathcal{A}$ as follows.

The simulation will be running as in the previous game $\mathbf{G_3}$ until the selected H_0 queries d_1, d_2, or d_3 are made. The simulator will abort the game if the inputs to three selected H_0 queries do *not* satisfy following conditions : (a) the passwords π_1, π_2, and π_3 are pair-wise distinct and different from the correct password π_C; and (b) the strings (C, S) in all three queries are the same.

If the selected H_0 queries are valid, $\langle X, Y, Z \rangle$ values will be plugged in according to the following rules:

⋆ **Rule** $H_0^{(4)}$

For the d_1th query set $r := X^{u_1}$. For the d_2th query set $r := Y^{u_2}$. For the d_3th query set $r := Z^{u_3}$. For all three selected queries set $\alpha = \perp$. Otherwise, choose $\alpha \leftarrow \mathbb{Z}_q$ and compute $r := g^{\alpha}$. In any case, write the record (w, r, α) to $\mathcal{L}_{h0}$. If $(*, r, *) \in \mathcal{L}_{\mathcal{A}}$, abort the game.

The prerequisites for the **Coll**$_C$ event to occur are: (1) valid d_1th, d_2th, or d_3th H_0 queries are selected by the simulator; (2) a pair (E_1, s_1) is generated by an honest client instance after a **Send**(C^i, **Start**) query; (3) the adversary generated a pair (E_2, s_2) and made a **Send**(C^i, (S, E_2, s_2)) query, where Good(E_2, s_2) is true and $E_2 \notin \{X, Y, Z\}$; (4a) for each PW_i, received from the selected H_0 queries, at least one bad $H_1(C, S, s_1, s_2, E_1, E_2, \sigma_i, PW_i)$ query is made for the same transcript, where $i \in \{1, 2, 3\}$. (4b) $\sigma_i \neq 1$.

After the game ends, for every $H_1(C, S, s_1, s_2, E_1, E_2, \sigma, PW_i)$ query $\mathcal{A}$ made, where PW_i is equal to any of the plugged values $\{X^{u_1}, Y^{u_2}, Z^{u_3}\}$, a pair

$$\left(E_2, (\sigma^{\frac{u_i}{\psi_1}} E_2^{\frac{-u_i s_1}{\psi_1}} PW_i^{\frac{-u_i s_1 s_2}{\psi_1}} g^{-u_i s_2}) \right) \tag{19}$$

is added to the list $\mathcal{L}^i_{bad}$.

So, in the case of an $H_1(C,\ S,\ s_1,\ s_2,\ E_1,\ E_2,\ \sigma,\ X^{u_1})$ query, by stripping away known values from σ, we may identify a guess at E_2^x and place it in the list $\mathcal{L}^1_{bad}$ together with the E_2 value. Remember that the client instance uses rule $\mathbf{C1}^{(4)}$ to compute E_1, which can be represented with $X^{u_1\psi_1 x}$. In order to extract the $F_1 = E_2^x$ value we do as follows. Since

$$\begin{aligned}\sigma &= DH_{X^{u_1}}(E_2 X^{u_1 s_2}, E_1 X^{u_1 s_1}) \\ &= E_2^{s_1} X^{u_1 s_1 s_2} E_2^{\frac{\psi_1 x}{u_1}} g^{s_2 \psi_1} \ ,\end{aligned} \tag{20}$$

we get

$$E_2^x = \sigma^{\frac{u_1}{\psi_1}} E_2^{\frac{-u_1 s_1}{\psi_1}} PW^{\frac{-u_1 s_1 s_2}{\psi_1}} g^{-u_1 s_2} \ . \tag{21}$$

The same goes for H_1 queries where the values Y and Z are plugged, in which case the corresponding F_2 and F_3 are computed, respectively. At the end of the simulation, $\mathcal{D}$ checks if for any E_2 value there exist pairs $(E_2, F_1) \in \mathcal{L}^1_{bad}$, $(E_2, F_2) \in \mathcal{L}^2_{bad}$ and $(E_2, F_3) \in \mathcal{L}^3_{bad}$, such that $F_1 F_2 = F_3$. If there exist three such pairs, then $\mathcal{D}$ will output $b' = 1$, and otherwise $b' = 0$.

Now let us analyze the probability that $\mathcal{D}$ returns a correct answer. Suppose first that Z is random. The algorithm $\mathcal{D}$ will return a wrong answer if by chance equation $F_1 F_2 = F_3$ holds. Since the u_i values are random, the probability of this happening is at most n^3_{h1}/q by the union bound. Now suppose that $Z = IDH_g(X, Y)$. The probability of aborting in case E_2 is equal to X, Y or Z is at most $3n_{se}/q$. If the adversary triggers $\mathbf{Coll}_C$, then $\mathcal{D}$ will correctly answer with $b' = 1$ only in case it correctly guessed d_1, d_2, and d_3 from $\{1, 2, \ldots, n_{h0}\}$, which happens with probability of $1/n^3_{h0}$. Therefore, the probability of $\mathcal{D}$ returning a correct answer is at least

$$\begin{aligned}\Pr[b' = b] &\geq \Pr[b' = 1 | b = 1]\Pr[b = 1] + \Pr[b' = 0 | b = 0]\Pr[b = 0] \\ &\geq \left(\frac{\Pr[\mathbf{Coll}_C]}{n^3_{h0}} - \frac{3n_{se}}{q}\right)\left(\frac{1}{2}\right) + \left(1 - \frac{n^3_{h1}}{q}\right)\left(\frac{1}{2}\right) \ .\end{aligned} \tag{22}$$

Thus,

$$\Pr[\mathbf{Coll}_C] \leq 2n^3_{h0} \times \left(\mathbf{Adv}^{didh}_{g,\mathbb{G}}(\mathcal{D}) + \frac{n^3_{h1} + 3n_{se}}{2q}\right) \ . \tag{23}$$

From the adversary's view, the simulation $\mathcal{D}$ runs is indistinguishable unless $\mathbf{Coll}_C$ event occurs. The probability of this happening is bounded by (23). $\mathcal{D}$'s running time is $t' = O(t + (n_{ex} + n_{ro} + n_{se})t_{exp})$ and thus Lemma 3 follows. □

Now, without any collision on the H_0 and H_1 outputs, $\mathcal{A}$ impersonating the server to an honest client instance can test at most two passwords per impersonation attempt. Therefore,

$$\Pr[\mathbf{AskH1\text{-}withC}_4] \leq \frac{2n_{se}}{N} \ . \tag{24}$$

Thus,

$$\Pr[\mathbf{AskH1}_4] \leq \frac{3n_{se}}{N} + \frac{n_{h1}^2}{2^{k+1}} + n_{h1}(1+n_{se}^2) \times Succ_{PW,\mathbb{G}}^{cdh}(t') + \Pr[\mathbf{Coll}_C] \ . \quad (25)$$

By combining the above equations the bound for semantic security follows. The bound for the mutual authentication is derived in a similar way, by noting that from Def. 2 we have $\mathbf{Adv}^{ma}_{\mathtt{dragonfly}}(\mathcal{A}) \leq \Pr[\mathbf{Auth}_0^{c2s}] + \Pr[\mathbf{Auth}_0^{s2c}]$. □

5 Conclusion

In this paper, using techniques similar to those MacKenzie used to study **SPEKE** in [17], we proved the security of a slight variant of **Dragonfly**, which gives some evidence that the IETF proposal of **Dragonfly** is sound. Furthermore, unlike the analysis of [17], we also include forward secrecy. (It is highly probable that **SPEKE** is forward secure as well.) Note that Theorem 1's statements indicate that the adversary may successfully guess up to six passwords per send query. Using a much more complex analysis, most likely we could replace the constant 6 in the non-negligible term by 2, and count per instance rather than per send query, but this would be at the cost of readability of the already intricate proof. Also, by virtue of the contents of Lemma 3, 2 is certainly the best we could do with this particular analysis.

It would also be nice to see if the proof can be made tighter. In particular, while it helps readability, the technique of using private oracles as in [7] seems less fine-grained than the systematic "backpatching" of, e.g. [18]. Finally, it would be interesting to see if the security of **Dragonfly** (and **SPEKE**) could be based on an assumption other than DIDH.

Acknowledgments. We thank the anonymous reviewers for their helpful comments. This work was partially supported by project SEQUOIA, a joint project between the *Fonds National de la Recherche, Luxembourg* and the *Agence Nationale de la Recherche* (France).

References

1. Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the J-PAKE Password-Authenticated Key Exchange Protocol. In: 2015 IEEE Symposium on Security and Privacy (2015), (Pages 6 and 11)
2. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure Against Dictionary Attacks. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer (2000)
3. Bellare, M., Rogaway, P.: Entity Authentication and Key Distribution. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO '93. LNCS, vol. 773, pp. 232–249. Springer (1993)
4. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: ACM Conference on Computer and Communications Security. pp. 62–73. ACM Press (1993)

5. Bellovin, S.M., Merritt, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: 1992 IEEE Computer Society Symposium on Research in Security and Privacy, May 4-6, 1992. pp. 72–84 (1992)
6. Boyko, V., MacKenzie, P.D., Patel, S.: Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer (2000)
7. Bresson, E., Chevassut, O., Pointcheval, D.: New Security Results on Encrypted Key Exchange. In: Bao, F., Deng, R.H., Zhou, J. (eds.) Public Key Cryptography. LNCS, vol. 2947, pp. 145–158. Springer (2004)
8. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally Composable Password-Based Key Exchange. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer (2005)
9. Clarke, D., Hao, F.: Cryptanalysis of the Dragonfly Key Exchange Protocol. Cryptology ePrint Archive, Report 2013/058 (2013), `http://eprint.iacr.org/`
10. Diffie, W., Hellman, M.: New Directions in Cryptography. IEEE Transactions on Information Theory 22(6), 644–652 (1976)
11. Harkins, D.: Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks. In: Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on. pp. 839–844 (Aug 2008)
12. Harkins, D.: Dragonfly Key Exchange (2015), `https://datatracker.ietf.org/doc/draft-irtf-cfrg-dragonfly/`
13. Jablon, D.P.: Strong Password-Only Authenticated Key Exchange. ACM SIGCOMM Computer Communication Review 26(5), 5–26 (1996)
14. Katz, J., Ostrovsky, R., Yung, M.: Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. LNCS, vol. 2045, pp. 475–494. Springer (2001)
15. Kwon, T.: Practical Authenticated Key Agreement Using Passwords. In: Zhang, K., Zheng, Y. (eds.) Information Security, LNCS, vol. 3225, pp. 1–12. Springer Berlin Heidelberg (2004)
16. Lancrenon, J., Skrobot, M.: On the Provable Security of the Dragonfly Protocol. In: Lopez, J., Mitchell, C.J. (eds.) Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings. LNCS, vol. 9290, pp. 244–261. Springer (2015)
17. MacKenzie, P.: On the Security of the SPEKE Password-Authenticated Key Exchange Protocol. Cryptology ePrint Archive, Report 2001/057 (2001), `http://eprint.iacr.org/2001/057`
18. MacKenzie, P.: The PAK Suite: Protocols for Password-Authenticated Key Exchange. DIMACS Technical Report 2002-46 (2002), (Page 7)
19. Pointcheval, D.: Password-Based Authenticated Key Exchange. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) Public Key Cryptography PKC 2012, LNCS, vol. 7293, pp. 390–397. Springer (2012)
20. Shoup, V.: On Formal Models for Secure Key Exchange. Cryptology ePrint Archive, Report 1999/012 (1999), `http://eprint.iacr.org/1999/012`

A The original Dragonfly protocol

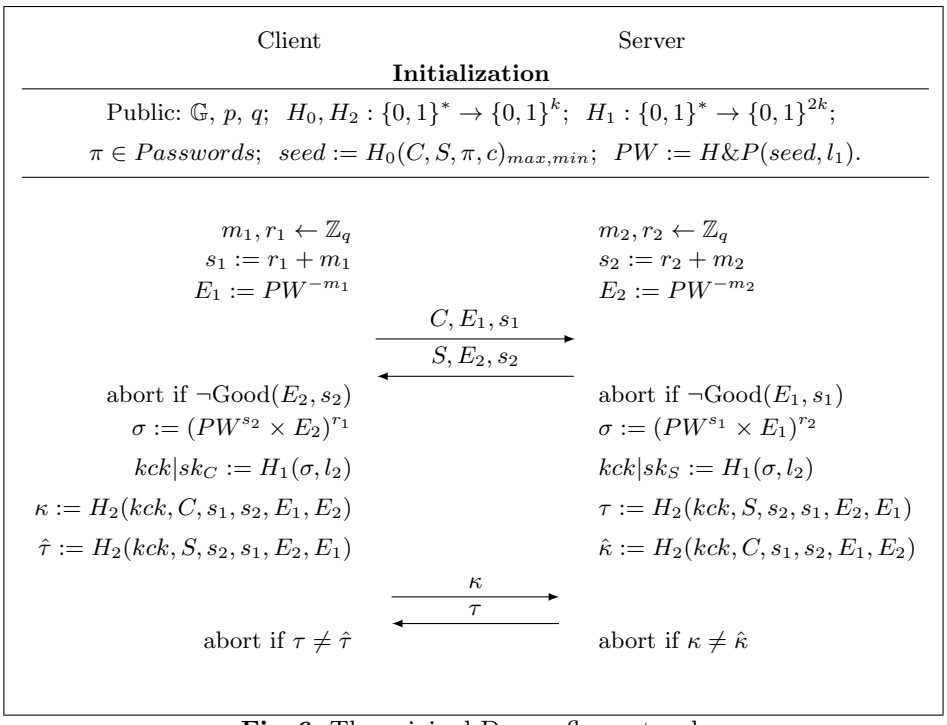

Fig. 6: The original Dragonfly protocol.