Tightly-Secure PAK(E)

José Becerra[1], Vincenzo Iovino[1], Dimiter Ostrev[1], Petra Šala[1][2], and Marjan Škrobot[1]

[1] University of Luxembourg
{name.lastname}@uni.lu,
[2] École Normale Supérieure, Computer Science Department

Abstract. We present a security reduction for the PAK protocol instantiated over Gap Diffie-Hellman Groups that is tighter than previously known reductions. We discuss the implications of our results for concrete security. Our proof is the first to show that the PAK protocol can provide meaningful security guarantees for values of the parameters typical in today's world.

Keywords: Password-Authenticated Key Exchange, PAK, Tight Reductions, Random Oracle.

1 Introduction

1.1 PAKE protocols

A password authenticated key exchange (PAKE) protocol allows two users who only share a password to establish a high entropy shared secret key by exchanging messages over a hostile network. PAKE protocols have only minimal requirements for the long-term secrets that users need to hold in order to succeed and therefore are interesting both theoretically and in practice. To date, there have been over twenty years of intensive research on PAKE, and PAKE protocols have recently seen more and more deployment in applications such as ad hoc networks [35] or the Internet of Things [32].

Numerous PAKE protocols have been proposed over the years. Among them, only a handful have been considered for use in real-world applications: EKE [6], SPEKE [17], SRP [36], PPK and PAK [8,26,25], KOY [19], Dragonfly [15], SPAKE2 [3] and J-PAKE [14]. The last two protocols, along with SRP and Dragonfly that have been standardized in the form of RFC2945 and RFC7664 respectively, are currently being considered by the Internet Engineering Task Force (IETF).

When evaluating different PAKE designs, two main criteria are the protocol's efficiency in terms of computation and communication, and the security guarantees that the protocol provides. Of these two criteria, the efficiency is easier to understand by just looking at the protocol description. On the other hand, it is difficult to judge whether a protocol is secure. A necessary condition for security is that no attacks on the protocol have been found so far, but most researchers agree that this is not sufficient.

1.2 Security models and reductions for PAKE

One way to rigorously discuss the security of PAKE protocols is to formally define a security challenge: an interaction between two algorithms called a challenger and an adversary. The interaction is designed to model the capabilities that a real world adversary is believed to have; the success of an adversary in the security challenge corresponds to a successful attack on the protocol. Several such security models have been introduced over the years. A few prominent ones are the indistinguishability-based models of Bellare, Pointcheval and Rogaway [5] and Abdalla, Fouque and Pointcheval [2], the simulation-based model of Boyko, MacKenzie and Patel [8][3], and the Universally Composable (UC) model of Canetti et al. [9].

In this approach, the security of a protocol is established in the following way: given an adversary $\mathcal{A}$ that runs in time t and has success probability ϵ in the security challenge, one constructs an algorithm $\mathcal{B}^{\mathcal{A}}$ known as a reduction. $\mathcal{B}^{\mathcal{A}}$ runs $\mathcal{A}$ as a subroutine and solves some known hard computational problem in time t' and with success probability ϵ'. If it is widely believed that it is impossible to solve the hard computational problem in time t' and with success probability ϵ', then one can conclude that no adversary running in time t can have a probability of ϵ to successfully attack the protocol.

1.3 Online dictionary attacks

Security models for PAKE must properly account for online dictionary attacks, in which an adversary guesses a password and tries to run the protocol with one of the honest users to verify the guess. Since passwords come from a small set, this attack has a non-negligible chance of success. Online dictionary attacks cannot be entirely prevented, but their effects can be mitigated to some extent for example by requiring users to choose strong passwords, limiting the number of unsuccessful login attempts, or even using machine learning to detect a pattern in login attempts that suggests an online dictionary attack might be in progress.

From the point of view of cryptographic research on PAKE, online dictionary attacks and the countermeasures listed above are taken as given; the focus is on ensuring that the adversary can do essentially no better than to run the best online dictionary attack in the circumstances. This intuitive requirement is formalized differently in indistinguishability-based and simulation-based models. In the indistinguishability-based model that we use in this paper[4], the formal requirement is that for all PPT adversaries $\mathcal{A}$ that perform at most n online dictionary attacks,

$$\mathrm{Adv}(\mathcal{A}) \leq \mathcal{F}(\mathcal{D}, \mathcal{L}, n) + \epsilon \tag{1}$$

[3] For the relation between the indistinguishability-based and simulation-based models, see the recent work [23].

[4] A detailed description of the FtG model of Bellare, Pointcheval and Rogaway [5] can be found in Section 4.

where $\mathrm{Adv}(\mathcal{A})$ is the advantage[5] of the adversary in breaking the protocol, where $\mathcal{F}(\mathcal{D}, \mathcal{L}, n)$ is the maximum probability of success[6] of any password guessing strategy that uses n guesses against a password distribution $\mathcal{D}$ and login attempt policy $\mathcal{L}$, and where ϵ is a negligible term.

In the present work, we focus on the behavior of the term ϵ of equation (1). A precise theoretical or empirical characterization of the function $\mathcal{F}(\mathcal{D}, \mathcal{L}, n)$ is an important and interesting research question, but is outside the scope of this paper. Here, we merely mention that many previous works use the formulation $\mathcal{F}(\mathcal{D}, \mathcal{L}, n) = n/N$ of [5,2] which corresponds to making the simplifying assumptions that there is no login attempt policy and that passwords are independent and uniformly distributed from a dictionary of size N. On the other hand, some recent research [33] suggests that real-life passwords follow Zipf's law, and proposes [34, Section 3] the formulation $\mathcal{F}(\mathcal{D}, \mathcal{L}, n) = Cn^s$ where C, s are parameters that have to be estimated empirically. Our results regarding the behavior of the term ϵ of equation (1) hold independently of the password distribution and login attempt policy, and in particular, they hold for any of the cases mentioned above.

1.4 The PAK protocol

One of the PAKE protocols whose security has been studied in the provable security framework is the PAK protocol [8,26,25]. It is a PAKE protocol with several desirable characteristics: low computation and communication cost, and security proofs in two different security models: the simulation-based model of Boyko, MacKenzie and Patel [8] and the so-called Find-then-Guess (FtG) model of Bellare, Pointcheval and Rogaway [5]. A modified version of PAK has been used to detect man-in-the-middle attacks against SSL/TLS without third-parties [10], and a lattice-based version of PAK has been used to provide security against quantum adversaries [11]. Moreover, variants of the PAK protocol have been included in IEEE standard [16], while the patent held by Lucent Technologies [27] is expiring soon. Therefore, the PAK protocol is a candidate for wide-scale practical deployment.

While there are security proofs for PAK in two different models, in both cases the reductions are loose, meaning either that the running time t' of the reduction $\mathcal{B}^{\mathcal{A}}$ is much larger than the running time t of the adversary $\mathcal{A}$ or that the success probability ϵ' of $\mathcal{B}^{\mathcal{A}}$ is much smaller than the success probability ϵ of $\mathcal{A}$, or some combination of the two.

A loose reduction is usually considered less than ideal. From a qualitative point of view, a reduction gives the assurance that "breaking the protocol is at most a little easier than solving the hard computational problem" [12]. However, if a reduction is loose, it leaves open the possibility that "a little easier" is in fact "substantially easier". From a quantitative point of view, a loose reduction means that larger security parameters must be chosen to guarantee a given level

[5] The advantage is twice the success probability minus one.

[6] By success we mean guessing the password of *any* user.

of security, which in turn increases the communication and computation cost of the protocol; therefore, a tight reduction is considered preferable [4].

We illustrate the last point by looking in detail at the best previous result for PAK [25, Theorem 6.9], which we reproduce here for convenience.

Theorem 1 (Theorem 6.9 in [25]). *Consider the PAK protocol*[7] *instantiated over a group $\mathbb{G} = \langle g \rangle$ of order q and with password dictionary of size N. Let $\mathcal{A}$ be an adversary that runs in time t and performs at most n_{se}, n_{ex}, n_{re}, n_{co}, n_{ro} queries of type* ***Send, Execute, Reveal, Corrupt, Random Oracle*** *and a single* ***Test*** *query. Let $\mathrm{Adv}(\mathcal{A})$ be the advantage of this adversary in the security challenge as defined in the FtG model*[8]*. Let t_{exp} be the time required for an exponentiation in $\mathbb{G}$. Then, for $t' = O(t + ((n_{ro})^2 + n_{se} + n_{ex})t_{exp})$*

$$\mathrm{Adv}(\mathcal{A}) = \frac{n_{se}}{N} + O\left(n_{se}\mathrm{Adv}_{\mathbb{G}}^{\mathtt{1\text{-}cdh}}(t', (n_{ro})^2) + \frac{(n_{se}+n_{ex})(n_{ro}+n_{se}+n_{ex})}{q}\right) \tag{2}$$

where $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{1\text{-}cdh}}(t', (n_{ro})^2)$ is the maximum success probability of an algorithm that is allowed to run for time t' and to output a list of $(n_{ro})^2$ candidate solutions to the CDH problem, and succeeds if at least one solution in the list is correct.

We plug in some concrete values in the above theorem. For the order of the group q, we use the recommended $q \approx 2^{256}$ for long-term security from [12, Chapter 7]. For the number of random oracle queries, we take $n_{ro} \approx 2^{63}$, the number of SHA1 computations performed in the recent attack [31]. Next, we use the approximation that solving the discrete logarithm problem in group $\mathbb{G}$ takes about $\sqrt{q} \approx 2^{128}$ operations [22, Section 7]. We see that with these values of the parameters, we can estimate

$$\mathrm{Adv}_{\mathbb{G}}^{\mathtt{1\text{-}cdh}}(t', (n_{ro})^2) \approx 1$$

and therefore the term

$$n_{se}\mathrm{Adv}_{\mathbb{G}}^{\mathtt{1\text{-}cdh}}(t', (n_{ro})^2) >> 1$$

makes the right hand-side of Eq.2 meaningless in bounding $\mathrm{Adv}(\mathcal{A})$, which, by definition, is a number less than or equal to 1. This means that we cannot reasonably claim that the security proof gives the guarantee "an adversary can essentially do no better than an online dictionary attack", except in the trivial case when the online dictionary attack itself succeeds with probability close to one.

1.5 Our contribution

We provide a tight reduction for PAK instantiated over Gap Diffie-Hellman groups; these are groups in which solving the Decisional Diffie-Hellman Problem

[7] A detailed description of the protocol is in Section 3.

[8] A detailed description of the FtG model of Bellare, Pointcheval and Rogaway [5] can be found in Section 4.

is easy but solving the Computational Diffie-Hellman problem is equivalent to solving the Discrete Logarithm Problem and is believed to be hard [18][9]. We employ proof techniques that have been used previously in [1,20].

The formal statement of our result can be found in Theorem 2.

Theorem 2. *Consider the PAK protocol instantiated over a Gap Diffie-Hellman group $\mathbb{G}_1 = \langle g \rangle$ of order q and with password dictionary of size N. Let $\mathcal{A}$ be an adversary that runs in time t and performs at most n_{se}, n_{ex}, n_{re}, n_{co}, n_{ro} queries of type **Send**, **Execute**, **Reveal**, **Corrupt**, **Random Oracle** and a single **Test** query. Let $\mathrm{Adv}(\mathcal{A})$ be the advantage of this adversary in the security challenge as defined in the FtG model. Let t_{exp} and t_{ddh} be the time required for an exponentiation in $\mathbb{G}_1$ and deciding DDH in $\mathbb{G}_1$, respectively. Then, for $t'' = O(t + (n_{ro} + n_{se} + n_{ex})t_{exp} + (n_{se} + n_{ro})t_{ddh})$*

$$\mathrm{Adv}(\mathcal{A}) \leq \mathcal{F}(\mathcal{D}, \mathcal{L}, n_{se}) + 8\mathrm{Adv}^{\mathsf{gap\text{-}cdh}}_{\mathbb{G}_1}(t'') + O\left(\frac{(n_{se} + n_{ex})(n_{ro} + n_{se} + n_{ex})}{q}\right) \tag{3}$$

where $\mathrm{Adv}^{\mathsf{gap\text{-}cdh}}_{\mathbb{G}_1}(t'')$ is the maximum success probability of an algorithm that is allowed to run for time t'' in solving the Gap-Diffie-Hellman (Gap-DH) problem in group $\mathbb{G}_1$.

We perform a similar analysis of our result as in the previous section, using the same values of q, and n_{ro}. Since $t'' << 2^{128}$ (assuming the most powerful adversaries today have t at most $\approx 2^{80}$ to 2^{85}), we can assume that $\mathrm{Adv}^{\mathsf{gap\text{-}cdh}}_{\mathbb{G}_1}(\approx 2^{85}) \lessapprox 2^{-35}$. Furthermore, the term $O\left((n_{se} + n_{ex})(n_{ro} + n_{se} + n_{ex})/q\right)$ is negligible compared to the other two terms. Thus, by using the tight reduction, we are able to obtain the guarantee: assuming that $\mathrm{Adv}^{\mathsf{gap\text{-}cdh}}_{\mathbb{G}_1}(\approx 2^{85}) \lessapprox 2^{-35}$ then for all adversaries $\mathcal{A}$ with running time $t \lessapprox 2^{85}$, the advantage in breaking the PAK protocol instantiated over a Gap Diffie Hellman group of order $\approx 2^{256}$ is at most $\approx 2^{-30}$ higher than the advantage of breaking the protocol using the best online dictionary attack for the given password distribution and login attempt policy.[10]

Thus, by relying on the Gap-Diffie-Hellman assumption instead of the List-Diffie-Hellman assumption as in [25] we are able to remove the degradation factors that cause the previous security proof for PAK to fail to provide meaningful guarantees for typical values of the parameters in today's world.

1.6 Organization of the paper

The rest of the paper is organized as follows: in Section 2, we introduce notation and give details on the Gap Diffie-Hellman groups and hardness assumptions

[9] More details on Gap Diffie-Hellman groups and the relevant computational problems and assumptions are given in Section 2.

[10] We refer to [34, Figure 4] for an estimation of the advantage of online dictionary attacks as a function of the number of guesses for two real-world password datasets.

used in this paper. In Section 3, we give a detailed description of the PAK protocol. In Section 4, we introduce the security model FtG of Bellare, Pointcheval and Rogaway [5]. In Section 5, we prove our main result. We conclude the paper in Section 6.

2 Preliminaries

In this section, we introduce notation, define pairings, and state the hardness assumptions upon which the security of PAK protocol rests.

2.1 Notation

We write $d \xleftarrow{\$} D$ for sampling uniformly at random from set D and $|D|$ to denote its cardinality. The output of a probabilistic algorithm A on input x is denoted by $y \leftarrow A(x)$, while $y := F(x)$ denotes a deterministic assignment of the value $F(x)$ to the variable y. Let $\{0,1\}^*$ denote the bit string of arbitrary length while $\{0,1\}^l$ stands for those of length l. Let κ be the security parameter and $negl(\kappa)$ denote a negligible function. When we sample elements from $\mathbb{Z}_q$, it is understood that they are viewed as integers in $[1 \ldots q]$, and all operations on these are performed $\mod q$. In general, we use $\mathbb{G}$ to denote any cyclic group while $\mathbb{G}_1$ refers to a bilinear group. Let H_1 be a full-domain hash mapping $\{0,1\}^*$ to $\mathbb{G}_1$. All remaining hash functions, H_2, H_3 and H_4, map from $\{0,1\}^*$ to $\{0,1\}^\kappa$.

2.2 Cryptographic building blocks

Let $\mathbb{G}_1, \mathbb{G}_T$ be cyclic groups of prime order q and g a generator of $\mathbb{G}_1$.

Definition 1. *A bilinear map is a function $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ such that the following properties are satisfied:*

1. *Bilinear: $\forall\ u, v \in \mathbb{G}_1,\ a, b \in \mathbb{Z}_q,\ e(u^a, v^b) = e(u, v)^{ab}$.*
2. *Non-degenerate: $e(g, g)$ generates $\mathbb{G}_T$.*
3. *Computable: $\forall\ u, v \in \mathbb{G}_1,\ a, b \in \mathbb{Z}_q$, there is an efficient algorithm to compute $e(u^a, v^b)$.*

Definition 2. *(Bilinear Group). $\mathbb{G}_1$ is a bilinear group if there exists group $\mathbb{G}_T$ and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$.*

2.3 Cryptographic hardness assumptions

Let $\mathbb{G}$ be any multiplicative cyclic group, with generator g and $|\mathbb{G}| = q$. For $X = g^x$ and $Y = g^y$, let $DH(X, Y) = g^{xy}$, where $\{g^x, g^y, g^{xy}\} \in \mathbb{G}$.

Definition 3. *(Computational Diffie-Hellman (CDH) Problem). Given $(g,\ g^x,\ g^y)$ compute g^{xy}, where $\{g^x, g^y, g^{xy}\} \in \mathbb{G}$ and $(x, y) \xleftarrow{\$} \mathbb{Z}_q^2$. Let the advantage of a PPT algorithm $\mathcal{A}$ in solving the CDH problem be:*

$$\mathrm{Adv}_{\mathbb{G}}^{\mathtt{cdh}}(\mathcal{A}) = \Pr\left[(x, y) \xleftarrow{\$} \mathbb{Z}_q^2, X = g^x, Y = g^y : \mathcal{A}(X, Y) = DH(X, Y)\right].$$

CDH assumption: There exist sequences of cyclic groups $\mathbb{G}$ indexed by κ such that for all PPT adversaries $\mathcal{A}$ $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{cdh}}(\mathcal{A}) \leq negl(\kappa)$, where κ is the security parameter.

Definition 4. *(List-Computational Diffie-Hellman (L-CDH) Problem). Given (g, g^x, g^y) compute g^{xy}, where $\{g^x, g^y, g^{xy}\} \in \mathbb{G}$ and $(x, y) \xleftarrow{\$} \mathbb{Z}_q^2$. Let $\mathcal{A}$ be a PPT algorithm which attempts to solve the L-CDH problem and outputs a list of n elements, its advantage is defined as follows:*

$$\mathrm{Adv}_{\mathbb{G}}^{\mathsf{l\text{-}cdh}}(\mathcal{A}, n) = \Pr\left[(x, y) \xleftarrow{\$} \mathbb{Z}_q^2, X = g^x, Y = g^y : DH(X, Y) \in \mathcal{A}(X, Y)\right].$$

L-CDH assumption: There exist sequences of cyclic groups $\mathbb{G}$ indexed by κ such that for all PPT adversaries $\mathcal{A}$ $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{l\text{-}cdh}}(\mathcal{A}, n) \leq negl(\kappa)$, where κ is the security parameter.

Definition 5. *(Decision Diffie-Hellman (DDH) Problem). Distinguish a tuple $(g^x,\ g^y,\ g^{xy})$ from (g^x, g^y, g^z), where $\{g^x, g^y, g^z\} \in \mathbb{G}_1$ and $(x, y, z) \xleftarrow{\$} \mathbb{Z}_q^3$. Let the advantage of a PPT algorithm $\mathcal{A}$ in solving DDH problem be:*

$$\begin{aligned}\mathrm{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{A}) = |\Pr\left[(x, y) \xleftarrow{\$} \mathbb{Z}_q^2, X = g^x, Y = g^y, Z = g^{xy} : \mathcal{A}(X, Y, Z) = 1\right] \\ - \Pr\left[(x, y, z) \xleftarrow{\$} \mathbb{Z}_q^3, X = g^x, Y = g^y, Z = g^z : \mathcal{A}(X, Y, Z) = 1\right]|. \quad (4)\end{aligned}$$

DDH assumption: There exist sequences of cyclic groups $\mathbb{G}$ indexed by κ such that for all PPT adversaries $\mathcal{A}$ $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{A}) \leq negl(\kappa)$, where κ is the security parameter.

Gap Diffie-Hellman (Gap-DH) groups are those where the DDH problem can be solved in polynomial time but no PPT algorithm can solve the CDH problem with advantage greater than negligible, e.g. bilinear groups from Def. 2. More formally:

Definition 6. *(Gap-Diffie-Hellman (Gap-DH) Problem). Given (g, g^x, g^y) and access to a Decision Diffie-Hellman Oracle (DDH-O) compute g^{xy}.*

$$\mathrm{Adv}_{\mathbb{G}_1}^{\mathsf{gap\text{-}cdh}}(\mathcal{A}) = \Pr\left[(x, y) \xleftarrow{\$} \mathbb{Z}_q^2, X = g^x, Y = g^y : \mathcal{A}^{ddh\text{-}o}(X, Y) = DH(X, Y)\right].$$

Gap-DH assumption: There exists sequences of *bilinear groups* $\mathbb{G}_1$ indexed by κ, such that for all PPT $\mathcal{A}$ $\mathrm{Adv}_{\mathbb{G}_1}^{\mathsf{gap\text{-}cdh}}(\mathcal{A}) \leq negl(\kappa)$, where κ is the security parameter.

3 The PAK Protocol

In this section, we describe the PAK protocol from [25], whose mathematical description is presented in Fig. 1. A few other variants of PAK were developed in [26].

3.1 Protocol description

Here, we make use of the same notation as in [25] (October version). Now, we describe the protocol informally.

Initialization

Public: $\mathbb{G}_1$, g, q; $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$;

$H_2, H_3, H_4 : \{0,1\}^* \rightarrow \{0,1\}^k$;

Client C		Server S
Secret: π		$\pi_S[C] = (H_1(\pi_C))^{-1}$
$x \xleftarrow{\$} \mathbb{Z}_q$		
$\alpha := g^x$		
$\gamma := H_1(\pi)$		
$m := \alpha \cdot \gamma$	$\xrightarrow{C, m}$	
		abort if $\neg acceptable(m)$
		$y \xleftarrow{\$} \mathbb{Z}_q$
		$\mu := g^y$
		$\gamma' := \pi_S[C]$
		$\sigma := (m \cdot \gamma')^y$
		$k := H_2(C, S, m, \mu, \sigma, \gamma')$
		$k'' := H_3(C, S, m, \mu, \sigma, \gamma')$
	$\xleftarrow{\mu, k}$	$sk := H_4(C, S, m, \mu, \sigma, \gamma')$
abort if $\neg acceptable(\mu)$		
$\sigma := \mu^x$		
$\gamma' := \gamma^{-1}$		
abort if $k \neq H_2(C, S, m, \mu, \sigma, \gamma')$		
$k' := H_3(C, S, m, \mu, \sigma, \gamma')$		
$sk := H_4(C, S, m, \mu, \sigma, \gamma')$	$\xrightarrow{k'}$	
		abort if $k' \neq k''$

Fig. 1: The PAK protocol.

Before any protocol execution, public parameters are fixed and passwords are shared between clients and servers during the initialization phase. More specifically, for efficiency reasons and security in case of password file compromise, servers only keep the inverse element of each password's hash value.

The PAK protocol consists of three message rounds. In the first message round, the client sends a group element m – generated by multiplying a random group element α with the mask γ (also a group element) that is derived from the shared password π – along with its ID to the server. In the second message round, upon receiving the message C, m, the server first checks with the *acceptable* function if the received value m is an element of $\mathbb{G}_1$. Then, it selects a random group element μ, removes the mask from the received m, and computes the shared secret σ, confirmation codes k, k', a session key sk and sets sid and pid values (thus accepting). Once all these values are computed, the server sends μ and k to the client. Upon receiving the second message, the client first checks if μ is valid group element. If so, it computes the shared secret and confirmation code k and checks the validity of the latter. If all checks are correct, the client computes his confirmation code k' and a session key sk, sets sid and pid values, and then it sends k' in the third message round and terminates. The server, once it receives value k' and checks its validity, also terminates.

3.2 Instantiating the protocol over Gap Diffie-Hellman groups

Gap-DH groups were introduced in the pioneering work of Boneh, Lynn and Shacham [7]. For instance, Gap-DH groups can be derived by the supersingular elliptic curve given by the equation $y^2 = x^3 + 2x \pm 1$ over the field $\mathbb{F}_{3^l}$. It can be seen that for some values of l the number of points in this curve divides $3^{6l} - 1$. The value 6 is called the multiplier that has to be neither too small for the CDH problem to be hard, nor too big for the Decision Diffie-Hellman Oracle (DDH-O) to be efficient. An example of DDH-O on this curve is the Weil pairing [30]. Gap-problems were also studied by Okamoto and Pointcheval [29].

In order to have efficient PAK execution, $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ must be an efficiently computable function. We point the reader to [26,28] for efficient implementations of H_1. Note that it is crucial for such an algorithm to run in constant time, otherwise timing attacks on a password are possible. For more details on pairings, we refer readers to [13].

4 Model

For our proof, we will use the well-known Find-then-Guess (FtG) security model from [5], which guarantees security against an adversary fully controlling the network, concurrent sessions, loss of session keys, as well as forward secrecy. Furthermore, the security model incorporates the essential requirements that PAKE protocols must satisfy: i) an eavesdropper adversary should not learn any information about the password and ii) an adversary can verify at most one password guess per protocol execution in an active attack.

In the FtG model, security is defined via a security experiment G_{ftg} played between a challenger $\mathcal{CH}_{ftg}$ and some adversary $\mathcal{A}$. The task of $\mathcal{CH}_{ftg}$ is to administrate the security experiment while keeping the appropriate secret information outside from $\mathcal{A}$'s view. Roughly speaking, $\mathcal{A}$ wins the security experiment if he is able to distinguish established session keys from random strings.

We will start by formally defining PAKE protocols. This will be followed by an in-depth description of the FtG security model.

PAKE protocol. A PAKE protocol can be represented as a pair of algorithms $(genPW, \mathrm{P})$, where $genPW$ is a password generation algorithm and P is the description of the protocol that specifies how honest parties behave. A $genPW$ algorithm takes as input a set of possible passwords $Passwords$, together with a probability distribution $\mathcal{P}$.

Participants and passwords. In the two-party PAKE setting, each principal U is either from a $Clients$ set or a $Servers$ set, both of which are finite, disjoint, nonempty sets. The set ID represents the union of $Clients$ and $Servers$. Furthermore, we assume that each client $C \in Clients$ possesses a password π_C, while on the other hand each server $S \in Servers$ holds a vector of the passwords of all clients $\pi_S := \langle \pi_C \rangle_{C \in Clients}$.

Protocol execution. P is a PPT algorithm that specifies reaction of principals to network messages. In a real scenario, each principal may run multiple executions of P with different users, thus in our model each principal is allowed an unlimited number of *instances* executing P in parallel. We denote with Π_i^U the i-th instance of a U principal. In some places, where distinction matters, we will denote client instances with Π_i^C and server instances by Π_j^S.

When assessing the security of P, we assume that the adversary $\mathcal{A}$ has complete control of the network. Practically, this means that principals solely communicate through the adversary that may consider delaying, reordering, modifying, dropping messages sent by honest principals or injecting messages of its choice in order to attack the protocol. Moreover, the adversary has access to instances of the principals through the game's interface (offered by the challenger). Thus, while playing the security game, $\mathcal{A}$ provides the inputs to the challenger $\mathcal{CH}_{ftg}$ – who parses the received messages and forwards them to corresponding instances – via the following *queries*:

– $\mathsf{Send}(U, i, M)$: $\mathcal{A}$ sends message M to instance Π_i^U. As a response, Π_i^U processes M according to the protocol description P, updates its corresponding *internal state* and outputs a reply that is given to $\mathcal{A}$. Whenever this query causes Π_i^U to *accept*, *terminate* or *abort*, it is indicated to $\mathcal{A}$. Additionally, to instruct client C to initiate a session with server S, the adversary sends a message containing the name of the server to an unused instance of C, i.e. $\mathsf{Send}(C, i, S)$.

– $\mathsf{Execute}(C, i, S, j)$: This triggers an honest run of P between instances Π_i^C and Π_j^S. The transcript of the protocol execution is given to $\mathcal{A}$. It covers passive eavesdropping on protocol flows.

– $\mathsf{Reveal}(U, i)$: As a response to this query, $\mathcal{A}$ receives the current value of the session key sk_U^i computed at Π_i^U. $\mathcal{A}$ may do this only if Π_i^U holds a session key, e.g. it is in *accept* or *terminate* state. This query captures potential session key leakage as a result of its use in higher level protocols. Also, it ensures that if some session key gets compromised, other session keys remain protected.

– **Test**(U, i): $\mathcal{CH}_{ftg}$ flips a bit b and answers this query as follows: if $b = 1$, $\mathcal{A}$ gets sk_U^i. Otherwise, it receives a random string from the session key space. This query can only be asked once by $\mathcal{A}$ at any time during the execution of G_{ftg}. This query simply measures the adversarial success and does not correspond to any real-world adversarial capability.

– **Corrupt**(U): The password π_U is given to $\mathcal{A}$ if U is a client, and the list of passwords π_U in case U is a server [11].

As can be seen above, the adversary is allowed to send multiple **Send**, **Execute**, **Reveal** and **Corrupt** queries to the challenger, and only a single **Test** query.

Accepting and terminating. In the FtG model from [5], an instance Π_i^U accepts whenever it holds a session key sk_U^i, a session ID sid_U^i and a partner ID pid_U^i. Note that the meaning of "accept" in this context is different from the usage of "accept" in other settings such as computational complexity.

An instance Π_i^U terminates if it holds sk_U^i, sid_U^i, pid_U^i and will not send nor receive any more messages. Due to the protocol design, Π_i^U may accept once and terminate later. Note also that it is possible for a server running the PAK protocol to accept at the time it sends the second protocol flow and to later abort if it receives a wrong confirmation code in the third protocol flow.

Partnering. We say that instances Π_i^C and Π_j^S are partnered if both oracles accept holding (sk_C^i, sid_C^i, pid_C^i) and (sk_S^j, sid_S^j, pid_S^j) respectively and the following conditions hold:

1. $sk_C^i = sk_S^j$, $sid_C^i = sid_S^j$, $pid_C^i = S$, $pid_S^j = C$
2. no other instance accepts with the same sid.

Freshness. It captures the idea that the adversary should not trivially know the session key being tested. We incorporate forward secrecy in the definition of freshness. An instance Π_i^U is said to be fs-fresh unless i) a **Reveal** query was made to Π_i^U or its partner (if it has one) or ii) a **Corrupt**(U') query was made before the **Test** query (where U' is any participant) and **Send**(U, i, M) query was made at some point.

PAKE security. The goal of $\mathcal{A}$ is to guess the bit b used to answer the test query. Let $\mathrm{Succ}_P^{\mathrm{FtG}}(\mathcal{A})$ be the event where $\mathcal{A}$ asks a single *test* query directed to a *fs-fresh* instance and $\mathcal{A}$ outputs his guess b', where $b' = b$. The advantage of $\mathcal{A}$ attacking P is defined as:

$$\mathrm{Adv}_P^{\mathrm{FtG}}(\mathcal{A}) = 2 \cdot \Pr\left[\mathrm{Succ}_P^{\mathrm{FtG}}(\mathcal{A})\right] - 1 \tag{5}$$

[11] This is the weak-corruption model of [5].

In the original formulation of the model from [5], we say that protocol P is FtG-secure if there exists a positive constant B such that for every PPT adversary $\mathcal{A}$ it holds that

$$\mathrm{Adv}_P^{\mathrm{FtG}}(\mathcal{A}) \leq \frac{B \cdot n_{se}}{N} + \varepsilon \tag{6}$$

where n_{se} is an upper bond on the number of Send queries $\mathcal{A}$ makes, and ε is negligible in the security parameter. Following our discussion in section 1.3, we can modify the definition to allow arbitrary password distribution and login attempt policy; thus, we can define a protocol to be secure if for for every PPT adversary $\mathcal{A}$,

$$\mathrm{Adv}_P^{\mathrm{FtG}}(\mathcal{A}) \leq \mathcal{F}(\mathcal{D}, \mathcal{L}, n_{se}) + \varepsilon \tag{7}$$

where we are using n_{se} as an upper bound on the number of online password guesses the adversary can make.
The following fact can be easily verified using Eq. 5:

Fact 1

$$\Pr\left[\mathrm{Succ}_P^{\mathrm{FtG}}(\mathcal{A})\right] = \Pr\left[\mathrm{Succ}_{P'}^{\mathrm{FtG}}(\mathcal{A})\right] + \epsilon \Leftrightarrow \mathrm{Adv}_P^{\mathrm{FtG}}(\mathcal{A}) = \mathrm{Adv}_{P'}^{\mathrm{FtG}}(\mathcal{A}) + 2\epsilon. \tag{8}$$

5 Proof of Security

In this section, we prove the security of the PAK protocol instantiated over Gap Diffie-Hellman groups. Due to similarity with the proof of the original PAK protocol [25], we present an overview for those security games that remain the same as in the original protocol and focus on those that deviate from the original proof. In Fig. 2 we provide the description of the game hops and highlight those games which differ from the original security proof. The terminology regarding adversary's actions, partnering and events stays as in [25] (see App. A).

$\mathbf{G_0}$: Original protocol.
$\mathbf{G_1}$: Force uniqueness of instances.
$\mathbf{G_2}$: Forbid lucky guesses on hash outputs and backpatch for consistency.
$\mathbf{G_3}$: **Randomize session keys for Execute queries (L-CDH).**
$\mathbf{G_4}$: Check password guesses.
$\mathbf{G_5}$: **Randomize session keys for paired instances (L-CDH).**
$\mathbf{G_6}$: **Forbid two password guesses per online attempt on server (L-CDH).**
$\mathbf{G_7}$: Internal password oracle.

Fig. 2: Description of games for the original PAK.

The main difference between the existing proof in the FtG model and our proof is that our reduction algorithm makes use of a Decisional Diffie-Hellman Oracle (DDH-O). Such oracle is available in gap groups, and it will output 1 on input (g, g^x, g^y, g^z) if $g^z = DH(g^x, g^y)$ and 0 otherwise. This additional information can be leveraged – in games G_3, G_5 and G_6 – to increase the success probability and reduce the running time of the reduction compared to Theorem 6.9 in [25].

Proof of Theorem 2: We will denote by P_i the protocol executed in game G_i, for i from 0 to 7. Before we start with the revised games, we will first describe in Fig. 3 how the random oracle queries to H_1 are answered by the simulator (reduction). It is important to highlight that the simulator has access to $\psi_1[\pi]$ values (see App. B).

Game $\mathbf{G_0}$: Original protocol. In this game, the challenger runs the original protocol P_0 for the adversary $\mathcal{A}$.

Game $\mathbf{G_1}$: Force uniqueness of instances. Let G_1 be exactly the same as G_0, except that if any of the values m and μ chosen by honest instances collide with previously generated ones, the protocol aborts and the adversary fails.

The probability of this event happening is negligible in the security parameter and limited by the birthday bound. More precisely, for all adversaries $\mathcal{A}$:

$$\mathrm{Adv}_{P_0}^{\mathrm{FtG}}(\mathcal{A}) \leq \mathrm{Adv}_{P_1}^{\mathrm{FtG}}(\mathcal{A}) + \frac{(n_{se}+n_{ex})(n_{se}+n_{ex}+n_{ro})}{q}. \tag{9}$$

Game $\mathbf{G_2}$: Forbid lucky guesses on hash outputs and backpatch for consistency. Let G_2 be the same as G_1, with the difference that now the simulator answers Send and Execute queries without making any random oracle queries, while ensuing random oracle queries are backpatched to ensure consistency in the view of the adversary.

In addition, G_2 forbids lucky guesses on hash functions. Specifically, in G_1 there are cases where an unpaired client instance Π_i^C may accept a confirmation code k, but the adversary has not asked the required random oracle queries to H_1 and H_2 in order to compute k, i.e. he proactively produced the correct one. The probability of this event happening is $\frac{\mathcal{O}(n_{ro}+n_{se})}{q}$. A similar scenario occurs when considering an unpaired server instance. Then:

$$\mathrm{Adv}_{P_1}^{\mathrm{FtG}}(\mathcal{A}) = \mathrm{Adv}_{P_2}^{\mathrm{FtG}}(\mathcal{A}) + \frac{\mathcal{O}(n_{ro}+n_{se})}{q}. \tag{10}$$

Game $\mathbf{G_3}$: Randomize session keys for Execute queries. Let G_3 be exactly the same as G_2, except that during processing of an $H_l(C, S, m, \mu, \sigma, \gamma')$ query for $l \in \{2, 3, 4\}$, there is no check for a $\mathbf{testexecpw}(C, i, S, j, \pi_c)$ event.

As a result of this change, even if $\mathbf{testexecpw}(C, i, S, j, \pi_c)$ event is triggered, the simulator will answer an $H_l(C, S, m, \mu, \sigma, \gamma')$ query with a random string from $\{0, 1\}^\kappa$.

Claim 1 *For all adversaries $\mathcal{A}$ running in time t, there exists an algorithm $\mathcal{D}$ running in time $t'' = \mathcal{O}(t + (n_{ro} + n_{se} + n_{ex}) \cdot t_{exp} + n_{ro} \cdot t_{ddh})$, such that:*

$$\mathrm{Adv}_{P_2}^{\mathrm{FtG}}(\mathcal{A}) \leq \mathrm{Adv}_{P_3}^{\mathrm{FtG}}(\mathcal{A}) + 2\mathrm{Adv}_{\mathbb{G}_1}^{\mathsf{gap\text{-}cdh}}(t''). \tag{11}$$

Proof: Let ϵ be the probability that **testexecpw** occurs in G_2. In that case $Pr(\mathrm{Succ}_{P_2}^{\mathrm{FtG}}(\mathcal{A})) \leq Pr(\mathrm{Succ}_{P_3}^{\mathrm{FtG}}(\mathcal{A})) + \epsilon$. By Fact 1, $\mathrm{Adv}_{P_2}^{\mathrm{FtG}}(\mathcal{A}) \leq \mathrm{Adv}_{P_3}^{\mathrm{FtG}}(\mathcal{A}) + 2\epsilon$. Note that games G_2 and G_3 are indistinguishable if **testexecpw** does not occur.

Now, we will construct an algorithm $\mathcal{D}$ that attempts to win its Gap-DH game against $\mathcal{CH}_{cdh}$ by running $\mathcal{A}$ as a subroutine on a simulation of the protocol P_2. For fixed (X, Y) that are coming from $\mathcal{CH}_{cdh}$, $\mathcal{D}$ simulates G_2 to $\mathcal{A}$ with the following changes:

1. For every **Execute**(C, i, S, j) query, set $m = X \cdot g^{\rho_{i_C}}$, $\mu = Y \cdot g^{\rho_{j_S}}$, where $(\rho_{i_C}, \rho_{j_S}) \xleftarrow{\$} \mathbb{Z}_q^2$, while k, k' stay random strings from $\{0,1\}^\kappa$.
2. Each time $\mathcal{A}$ asks a $H_l(C, S, m, \mu, \sigma, \gamma')$ query for $l \in \{2, 3, 4\}$ – where values m, μ were generated in **Execute**(C, i, S, j) query, and $H_1(\pi_c)$ query returned $(\gamma')^{-1}$ – $\mathcal{D}$ calls DDH-O with input $(m \cdot \gamma', \mu, \sigma)$. Once DDH-O returns 1, the "winning" H_l query is identified, and $\mathcal{D}$ computes Z value as follows

$$Z = \sigma \cdot X^{\rho_{j_S}} \cdot Y^{\rho_{i_C}} \cdot g^{\rho_{i_C} \cdot \rho_{j_S}} \cdot \mu^{\psi_1[\pi_c]}, \tag{12}$$

submits it to $\mathcal{CH}_{cdh}$ as a solution for (X, Y) challenge, and stops. The advantage of $\mathcal{D}$ in solving Gap-DH is equal to ϵ and its running time is $t'' = \mathcal{O}(t + (n_{se} + n_{ex} + n_{ro})t_{exp} + n_{ro}t_{ddh})$. □

DISCUSSION. Notice that the running time of $\mathcal{D}$ in G_3 has slightly increased (by $n_{ro}t_{ddh} - n_{ro}t_{exp}$) when comparing with MacKenzie's reduction, since $c \cdot t_{exp} = t_{ddh}$, where c is some constant. As in [25], $\epsilon' = \epsilon$. However, here only a single Z value is computed and sent, in contrast to the existing reduction where a list of size n_{ro} is submitted to $\mathcal{CH}_{cdh}$.

Game $\mathbf{G_4}$: Check password guesses. The challenger executes P_3 as in G_3, except that if **correctpw** event occurs, then the protocol execution aborts and the adversary succeeds.

As consequence, before any **Corrupt** query, whenever the simulator detects (via oracle queries) that the adversary uses the correct password to compute the confirmation code k, the protocol will be aborted and the adversary will be deemed successful, i.e., no *unpaired* client or server instance will terminate prior to **correctpw** event or **Corrupt** query.

$$\mathrm{Adv}_{P_3}^{\mathrm{FtG}}(\mathcal{A}) \leq \mathrm{Adv}_{P_4}^{\mathrm{FtG}}(\mathcal{A}). \tag{13}$$

Game $\mathbf{G_5}$: Randomize session keys for paired instances. G_5 is identical to G_4, except in case **pairedpwguess** event occurs. In that case, the game stops and adversary fails.

In this particular reduction, we will show that an adversary $\mathcal{A}$ who i) can adaptively corrupt user (thus knowing the password π_c) and ii) manages to compute *sk* for *paired* instances $\mathit{\Pi}_i^C$ and $\mathit{\Pi}_j^S$, could be used as a subroutine to solve the Gap-DH problem.

Claim 2 *For any adversary $\mathcal{A}$ running in time t, an algorithm $\mathcal{D}$ running in time $t'' = \mathcal{O}(t + (n_{se} + n_{ro} + n_{exe})t_{exp} + (n_{se} + n_{ro})t_{ddh})$ can be built such that:*

$$\mathrm{Adv}_{P_4}^{\mathrm{FtG}}(\mathcal{A}) \leq \mathrm{Adv}_{P_5}^{\mathrm{FtG}}(\mathcal{A}) + 2\mathrm{Adv}_{\mathbb{G}_1}^{\mathsf{gap\text{-}cdh}}(t''). \tag{14}$$

Proof: If **pairedpwguess** does not occur, then games G_4 and G_5 are indistinguishable. Let ϵ be the probability that **pairedpwguess** event occurs, when $\mathcal{A}$ is running in G_4.

Next, we will construct an algorithm $\mathcal{D}$ that attempts to win its Gap-DH game against $\mathcal{CH}_{cdh}$ by running $\mathcal{A}$ as a subroutine on a simulation of the protocol P_4. For a given pair (X, Y), $\mathcal{D}$ simulates G_4 to $\mathcal{A}$ with the following changes:

1. In CLIENT ACTION 0 query to Π_i^C and input S, set $m = X \cdot g^{\rho_{C,i}}$ where $\rho_{C,i} \xleftarrow{\$} \mathbb{Z}_q$.
2. In SERVER ACTION 1 query to Π_j^S and input $\langle C, m\rangle$, set $\mu = Y \cdot g^{\rho_{S,j}}$, where $\rho_{S,j} \xleftarrow{\$} \mathbb{Z}_q$.
3. In CLIENT ACTION 1 to Π_i^C and input $\langle \mu, k\rangle$, if Π_i^C is unpaired, $\mathcal{D}$ first verifies k using DDH-O and the list of random oracle queries. If k is correctly constructed (DDH-O outputs 1), Π_i^C outputs k' and terminates, or rejects otherwise.
4. In SERVER ACTION 2 query to Π_j^S with input k', if Π_j^S was paired after its SERVER ACTION 1 but is now unpaired, then $\mathcal{D}$ verifies k'. If k' is correctly constructed, then Π_j^S terminates. Otherwise, it rejects.
5. After $\mathcal{A}$ terminates, the simulator selects queries of the form $H_l(C, S, m, \mu, \sigma, \pi)$, for which the following conditions are satisfied: i) m and μ generated by some instances Π_i^C and Π_j^S respectively, ii) Π_i^C is paired with Π_j^S and Π_j^S is paired with Π_i^C after SERVER ACTION 1, iii) $(\gamma')^{-1} = H_1(\pi)$. For every such query, $\mathcal{D}$ calls DDH-O with input $(m \cdot \gamma', \mu, \sigma)$.

Once DDH-O returns 1, $\mathcal{D}$ computes Z value in the same way as for G_3 (Eq. 12), submits it to $\mathcal{CH}_{cdh}$ as a solution for (X, Y) challenge, and stops. The advantage of $\mathcal{D}$ in solving Gap-DH is equal to ϵ and its running time is $t'' = \mathcal{O}(t + (n_{se} + n_{ro} + n_{exe})t_{exp} + (n_{se} + n_{ro})t_{ddh})$. □

DISCUSSION. To explain why the original reduction from [25] contains n_{se} degradation factor, and how we can avoid such degradation in ours, consider the following scenario:

Suppose that the adversary $\mathcal{A}$ against protocol P_4 first makes a CLIENT ACTION 0 query to Π_i^C and receives as an answer $m = X \cdot g^{\rho_{C,i}}$ value in which Diffie-Hellman challenge X is planted. Next, $\mathcal{A}$ obtains $\pi_c = \pi[C, S]$ via **Corrupt**(S) query. With this information, $\mathcal{A}$ may decide to impersonate S to C by making a CLIENT ACTION 1 query with an input $\langle \mu, k\rangle$ to Π_i^C. Since $\mathcal{A}$ knows the correct password, he could compute and send the correct confirmation code k; however, $\mathcal{A}$ could also choose to send an incorrect one. Now, the simulator faces a problem: Π_i^C has to verify k and based on the verification outcome either accept or reject. Put differently, the simulator is unable to verify whether **testpw**$(C, i, S, \pi_c, l = 2)$ is triggered; this could be done by checking if $\sigma = DH(\alpha, \mu)$, but the simulator does not know the discrete log of X.

To circumvent this obstruction, the reduction in [25] has to guess an instance that will be the target of the **Test** query: this provides guarantee that there won't be any corruption before session keys are accepted, and thus the simulator can safely plant the received Diffie-Hellman challenge (X, Y) in the **Test** session. This technique yields a factor of n_{se} in front of $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{1\text{-}cdh}}$ advantage in Theorem 1.

In contrast, by using Gap-DH groups, our simulator can query DDH-O with input (α, μ, σ) to verify if $\sigma = DH(\alpha, \mu)$ and check whether the event **testpw**$(C, i, S, \pi_c, l = 2)$ is triggered or not. Hence, we can avoid guessing of the **Test** instance, which makes our reduction tight with respect to the success probability. Compared to [25], the running time of the reduction algorithm has increased by an additive term $(n_{se} + n_{ro})t_{ddh}$, due to the invocation of DDH-O needed for the simulator to identify correct random oracle queries.

Game $\mathbf{G_6}$: Forbid two password guesses per online attempt on server. Let G_6 be identical to G_5, except that if **doublepwserver** event occurs, the protocol halts and the adversary fails. We assume that the check for **doublepwserver** occurs before the check for **pairedpwguess**.

Claim 3 *For any adversary $\mathcal{A}$ running in time t, there exists an algorithm $\mathcal{D}$ running in time $t'' = \mathcal{O}(t + (n_{se} + n_{ro} + n_{exe})t_{exp} + n_{ro}t_{ddh})$ such that*

$$\mathrm{Adv}_{P_5}^{\mathrm{FtG}}(\mathcal{A}) \leq \mathrm{Adv}_{P_6}^{\mathrm{FtG}}(\mathcal{A}) + 4\mathrm{Adv}_{\mathbb{G}_1}^{\mathsf{gap\text{-}cdh}}(t'') \tag{15}$$

Proof: We will construct an algorithm $\mathcal{D}$ that attempts to win its Gap-DH game against $\mathcal{CH}_{cdh}$ by running $\mathcal{A}$ as a subroutine on a simulation of the protocol P_5. For a given pair (X, Y), $\mathcal{D}$ simulates G_5 to $\mathcal{A}$ with the following changes:

1. In $H_1(\pi)$ query, output $X^{\psi_1[\pi]} g^{\psi_1'[\pi]}$, where $\psi_1[\pi] \xleftarrow{\$} \{0, 1\}$ and $\psi_1'[\pi] \xleftarrow{\$} \mathbb{Z}_q$.
2. In a SERVER ACTION 1 query to a server Π_j^S with input $\langle C, m\rangle$ where $acceptable(m)$ is true, set $\mu = Y \cdot g^{\rho'_{S,j}}$.
3. Tests for **correctpw** and **pairedpwguess**, from G_4 and G_5 respectively, are not made.
4. After $\mathcal{A}$ terminates, the simulator $\mathcal{D}$ using DDH-O first creates a list L_c of $H_l(C, S, m, \mu, \sigma, \gamma')$ queries, with $l \in \{2, 3, 4\}$, such that $\sigma = DH(m \cdot \gamma', \mu)$. Then $\mathcal{D}$ selects from the list L_c two different queries, say $H_l(C, S, m, \mu, \sigma, \gamma')$ and $H_{\hat{l}}(C, S, m, \mu, \hat{\sigma}, \hat{\gamma}')$, for $l, \hat{l} \in \{2, 3, 4\}$ such that there was i) a SERVER ACTION 1 query to a server instance Π_j^S with input $\langle C, m\rangle$ and output $\langle \mu, k\rangle$, ii) an $H_1(\pi)$ query that returned $(\gamma')^{-1}$, an $H_1(\hat{\pi})$ query that returned $(\hat{\gamma}')^{-1}$ and iii) $\psi_1[\pi] \neq \psi_1[\hat{\pi}]$. Then $\mathcal{D}$ outputs:

$$Z = \left(\sigma \cdot \hat{\sigma}^{-1} \cdot (\gamma')^{-\rho'_{S,j}} \cdot (\hat{\gamma}')^{\rho'_{S,j}} \cdot Y^{\psi_1'[\pi] - \psi_1'[\hat{\pi}]}\right)^{\psi_1[\pi] - \psi_1[\hat{\pi}]}, \tag{16}$$

where $Z = DH(X, Y)$.

G$_6$ is indistinguishable from G$_5$ until the event **doublepwserver** occurs. Let the ϵ be the probability that **doublepwserver** occurs when $\mathcal{A}$ is running in G$_5$. When **doublepwserver** occurs for two passwords $\pi \neq \hat{\pi}$, the success probability of $\mathcal{D}$ is $\epsilon/2$ and its running time is $t'' = \mathcal{O}(t + (n_{se} + n_{ro} + n_{exe})t_{exp} + n_{ro}t_{ddh})$. $\square$

DISCUSSION: This game shows that $\mathcal{A}$'s probability of simultaneously guessing (discarding) more than *one* password during a single online attempt on a server executing P$_6$ is negligible. In most PAKE proofs (in [25] too), this reduction typically brings the highest security degradation: e.g. $1/n_{ro}^3$ appears in the case of Dragonfly [21] and SPEKE [24]. In contrast, our protocol only suffers from a constant loss (4) in the success probability.

The reason for $1/n_{ro}^2$ degradation when using L-CDH in PAK reduction is the following: $\mathcal{D}$ has to compute and output a list of *possible* DH values and he expects the solution for CDH to be contained within the list if $\mathcal{A}$ wins its game. The list is computed as follows: for particular pairs of queries $H_l(C, S, m, \mu, \sigma, \gamma')$ and $H_{\hat{l}}(C, S, m, \mu, \hat{\sigma}, \hat{\gamma}')$, for $l, \hat{l} \in \{2, 3, 4\}$, $\mathcal{D}$ computes Z as in Eq. 16 and adds it to his list of *possible* DH values. The size of the list is upper bounded by $(n_{ro})^2$, resulting in unfeasible running time for $\mathcal{D}$.

In contrast, by using Gap-DH groups, $\mathcal{D}$ can identify the right pair of H_l queries (at the cost of at most $n_{ro}t_{ddh}$ in the running time) and then compute a single, correct Z value using Eq. 16. As a result, we can remove the quadratic factor in the running time of $\mathcal{D}$.

Game G$_7$: **Internal password oracle.** The purpose of this game is to estimate the probability of the **correctpw** event occurring, i.e. the adversary guessing the correct password π_c.

Let G$_7$ be as G$_6$, except that there is an *internal password oracle* $\mathcal{O}_{pw}$ which generates all passwords during the initialization of the users. The simulator uses it to i) handle **Corrupt** queries and ii) test whether **correctpw** occurs. More specifically, when $\mathcal{A}$ asks **Corrupt**(U), the query is simply forwarded to $\mathcal{O}_{pw}$ which returns π_U if $U \in \mathit{Clients}$, otherwise returns $\langle \pi_U[C] \rangle_{C \in \mathit{Clients}}$. To determine whether **correctpw** occurs, the simulator queries $\mathcal{O}_{pw}$ with test(π, C), which returns TRUE if $\pi = \pi_C$ and FALSE otherwise.

By definition G$_6$ and G$_7$ are perfectly indistinguishable. Then:

$$\mathrm{Adv}_{P_6}^{\mathrm{FtG}}(\mathcal{A}) = \mathrm{Adv}_{P_7}^{\mathrm{FtG}}. \tag{17}$$

Claim 4 *For all PPT adversaries $\mathcal{A}$:*

$$\mathrm{Adv}_{P_7}^{\mathrm{FtG}} \leq \mathcal{F}(\mathcal{D}, \mathcal{L}, n_{se}). \tag{18}$$

Proof: Let ψ denote the **correctpw** event and ψ^c its compliment. The probability that $\mathcal{A}$ succeeds in G$_7$ is given by:

$$\begin{aligned} \Pr[\mathrm{Succ}_{P_7}^{\mathrm{FtG}}(\mathcal{A})] = \Pr[\psi] \cdot \Pr[\mathrm{Succ}_{P_7}^{\mathrm{FtG}}(\mathcal{A}) \mid \psi] +& \\ \Pr[\psi^c] \cdot \Pr[\mathrm{Succ}_{P_7}^{\mathrm{FtG}}(\mathcal{A}) \mid \psi^c]& \end{aligned} \tag{19}$$

We look at the first term of Eq. 19. Since there are n_{se} **Send** queries, the probability of **correctpw** occurring is bounded by $\Pr[\psi] \leq \mathcal{F}(\mathcal{D}, \mathcal{L}, n_{se})$. Additionally, it follows from G_4 that $\Pr[\mathrm{Succ}^{\mathrm{FtG}}_{P_7}(\mathcal{A}) \mid \psi] = 1$. Now we look at the second term of Eq. 19. Given that **correctpw** does not occur, $\mathcal{A}$ succeeds by making a **Test** query to a fresh instance Π_i^U and guessing the bit b used in the **Test** query. By examining **Reveal** and H_4 queries throughout the proof, it follows that the view of $\mathcal{A}$ is independent of sk_U^i, therefore $\Pr[\mathrm{Succ}^{\mathrm{FtG}}_{P_7}(\mathcal{A}) \mid \psi^c] = 1/2$. Putting everything together, using Eq. 19 and Eq. 5:

$$\mathrm{Adv}^{\mathrm{FtG}}_{P_7} \leq \mathcal{F}(\mathcal{D}, \mathcal{L}, n_{se}). \qquad \square$$

6 Conclusion

In this paper, we proposed a new instantiation for the PAK protocol and showed that the security proof from [25] can be adapted to cover our proposal. Our reduction to the Gap Diffie-Hellman problem is significantly tighter than the previous reduction to the List Diffie-Hellman problem. From a theoretical point of view, this shows that the security of PAK is closely related to the security of Gap-DH assumption. In terms of concrete security, the advantage of the tighter proof is that it provides the guarantee that with typical values of the group size for today, even the most computationally powerful adversaries today cannot do significantly better than an online dictionary attack. In future work it would be interesting to see if similar techniques could lead to tighter security proofs in other existing PAKE protocols.

Assumption	$t' - t - t_{sim}$	ϵ'/ϵ
Standard CDH	$\mathcal{O}(c \cdot t_{exp})$	$1/n_{ro}^2$
L-CDH	$\mathcal{O}((n_{ro})^2 \cdot t_{exp})$	$1/n_{se}$
Gap-DH	$\mathcal{O}(n_{ro} \cdot t_{ddh})$	1

Table 1. Comparison of running time and success probability of PAK reduction algorithm when using different variants of CDH assumption. Variable t_{exp} represents the running time to compute exponentiation in $\mathbb{G}$, t_{ddh} the time for deciding DDH, $t_{sim} = (n_{se} + n_{ro} + n_{exe})t_{exp}$, c is a constant, n_{ro} and n_{se} are the number of random oracle and send queries respectively.

ACKNOWLEDGEMENTS

We would like to thank the anonymous referees for their comments. This work was supported by the Luxembourg National Research Fund (CORE project AToMS and CORE Junior grant no. 11299247).

References

1. Abdalla, M., Chevassut, O., Pointcheval, D.: One-Time Verifier-Based Encrypted Key Exchange. In: Vaudenay, S. (ed.) Public Key Cryptography - PKC 2005. LNCS, vol. 3386, pp. 47–64. Springer (2005)
2. Abdalla, M., Fouque, P., Pointcheval, D.: Password-Based Authenticated Key Exchange in the Three-Party Setting. In: Vaudenay, S. (ed.) Public-Key Cryptography – PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer (2005)
3. Abdalla, M., Pointcheval, D.: Simple Password-Based Encrypted Key Exchange Protocols. In: Menezes, A. (ed.) Topics in Cryptology - CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer (2005)
4. Bellare, M.: Practice-Oriented Provable Security. In: Damgård, I. (ed.) Lectures on Data Security, Modern Cryptology in Theory and Practice. LNCS, vol. 1561, pp. 1–15. Springer (1998)
5. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure Against Dictionary Attacks. In: Advances in Cryptology – EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer (2000)
6. Bellovin, S.M., Merritt, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: 1992 IEEE Symposium on Research in Security and Privacy, SP 1992. pp. 72–84 (1992)
7. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. In: Boyd, C. (ed.) Advances in Cryptology - ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer (2001)
8. 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)
9. 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)
10. Dacosta, I., Ahamad, M., Traynor, P.: Trust No One Else: Detecting MITM Attacks against SSL/TLS without Third-Parties. In: Foresti, S., Yung, M., Martinelli, F. (eds.) Computer Security - ESORICS 2012. LNCS, vol. 7459, pp. 199–216. Springer (2012)
11. Ding, J., Alsayigh, S., Lancrenon, J., RV, S., Snook, M.: Provably Secure Password Authenticated Key Exchange Based on RLWE for the Post-Quantum World. In: Handschuh, H. (ed.) Topics in Cryptology - CT-RSA 2017. LNCS, vol. 10159, pp. 183–204. Springer (2017)
12. Ecrypt, I.: ECRYPT II Yearly Report on Algorithms and Keysizes. European Network of Excellence in Cryptology II, Tech. Rep (2012)
13. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for Cryptographers. Discrete Appl. Math. 156(16), 3113–3121 (Sep 2008)
14. Hao, F., Ryan, P.: J-PAKE: Authenticated Key Exchange without PKI. Transactions on Computational Science 11, 192–206 (2010)
15. Harkins, D.: Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks. In: Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications. pp. 839–844. SENSORCOMM '08, IEEE Computer Society (2008)
16. Standard Specifications for Password-Based Public Key Cryptographic Techniques. Standard, IEEE Standards Association, Piscataway, NJ, USA (2002)

17. Jablon, D.P.: Strong Password-Only Authenticated Key Exchange. ACM SIGCOMM Computer Communication Review 26(5), 5–26 (1996)
18. Joux, A., Nguyen, K.: Deparating Decision Diffie–Hellman from Computational Diffie–Hellman in Cryptographic Groups. Journal of Cryptology 16(4), 239–247 (2003)
19. 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)
20. Krawczyk, H.: Hmqv: A High-Performance Secure Diffie-Hellman Protocol. In: Shoup, V. (ed.) Advances in Cryptology - CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer (2005)
21. Lancrenon, J., Škrobot, M.: On the Provable Security of the Dragonfly Protocol. In: Lopez, J., Mitchell, C.J. (eds.) Information Security – ISC 2015. LNCS, vol. 9290, pp. 244–261. Springer (2015)
22. Lenstra, A.K.: Key Lengths. Tech. rep., Wiley (2006)
23. Lopez Becerra, J.M., Iovino, V., Ostrev, D., Škrobot, M.: On the Relation Between SIM and IND-RoR Security Models for PAKEs. In: SECRYPT 2017. SCITEPRESS (2017)
24. 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
25. MacKenzie, P.: The PAK Suite: Protocols for Password-Authenticated Key Exchange. DIMACS Technical Report 2002-46 (2002)
26. MacKenzie, P.D.: More Efficient Password-Authenticated Key Exchange. In: Naccache, D. (ed.) Topics in Cryptology - CT-RSA 2001. LNCS, vol. 2020, pp. 361–377. Springer (2001)
27. MacKenzie, P.: Methods and Apparatus for Providing Efficient Password Authenticated Key Exchange (2002), `https://www.google.com/patents/US20020194478`, publication number US20020194478 A1
28. Mrabet, N.E., Joye, M.: Guide to Pairing-Based Cryptography. Chapman & Hall/CRC (2016)
29. Okamoto, T., Pointcheval, D.: The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes. In: Kim, K. (ed.) Public-Key Cryptography - PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer (2001)
30. Silverman, J.H.: The Arithmetic of Elliptic Curves, vol. 106. Springer Science & Business Media (2009)
31. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The First Collision for full SHA-1. IACR Cryptology ePrint Archive 2017, 190 (2017), `http://eprint.iacr.org/2017/190`
32. Thread-Group: Thread Protocol. `http://threadgroup.org/` (2015)
33. Wang, D., Cheng, H., Wang, P., Huang, X., Jian, G.: Zipf's Law in Passwords. IEEE Transactions on Information Forensics and Security (2017)
34. Wang, D., Wang, P.: On the Implications of Zipf's Law in Passwords. In: Askoxylakis, I.G., Ioannidis, S., Katsikas, S.K., Meadows, C.A. (eds.) Computer Security - ESORICS 2016. LNCS, vol. 9878, pp. 111–131. Springer (2016)
35. Warner, B.: Magic Wormhole. `https://github.com/warner/magic-wormhole` (2016)
36. Wu, T.D.: The Secure Remote Password Protocol. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 1998. The Internet Society (1998)

A Terminology from the original proof of PAK

First, we introduce the terminology from [25] that deals with adversary's actions and partnering.

We say "in a CLIENT ACTION κ query to Π_i^C", to refer to "in a Send query to Π_i^C that results in execution of CLIENT ACTION κ procedure" and "in a SERVER ACTION κ query to Π_j^S", to refer to "in a Send query to Π_j^S that results in execution of SERVER ACTION κ procedure". A client instance Π_i^C is paired with a server instance Π_j^S if there is a CLIENT ACTION 0 query to Π_i^C with input S and output $\langle C, m\rangle$, there is a SERVER ACTION 1 query to Π_j^S with input $\langle C, m\rangle$ and output $\langle \mu, k\rangle$ and there is a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k\rangle$. A server instance Π_j^S *is paired* with client instance Π_i^C whenever there is a CLIENT ACTION 0 query to Π_i^C with input S and output $\langle C, m\rangle$, there is a SERVER ACTION 1 query to Π_j^S with input $\langle C, m\rangle$ and output $\langle \mu, k\rangle$, and if there is a SERVER ACTION 2 query to Π_j^S with input k', then there was previously a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k\rangle$ and output k'.

Next we describe those events taken from [25] which are required in our proof of security.

- **testpw**(C, i, S, π, l): for some m, μ and γ', $\mathcal{A}$ makes i) an $H_l(C, S, m, \mu, \sigma, \gamma')$ query, ii) a CLIENT ACTION 0 query to a client instance Π_i^C with input S and output $\langle C, m\rangle$, iii) a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k\rangle$ and iv) an $H_1(\pi)$ query returning $(\gamma')^{-1}$, where the last query is either the $H_l(\cdot)$ query or the CLIENT ACTION 1 query, $\sigma = DH(\alpha, \mu)$, $m = \alpha \cdot (\gamma')^{-1}$ and $l \in \{2, 3, 4\}$.
- **testpw!**(C, i, S, π): for some k, a CLIENT ACTION 1 query with input $\langle \mu, k\rangle$ causes a testpw$(C, i, S, \pi, 2)$ event to occur, with associated value k.
- **textpw**(S, j, C, π, l): for some m, μ, γ' and k, $\mathcal{A}$ makes an $H_l(C, S, m, \mu, \sigma, \gamma')$ query, and previously made i) a SERVER ACTION 1 query to a server instance Π_j^S with input $\langle C, m\rangle$ and output $\langle \mu, k\rangle$, and ii) an $H_1(\pi)$ query returning $(\gamma')^{-1}$, where $\sigma = DH(\alpha, \mu)$, $m = \alpha \cdot (\gamma')^{-1}$ and ACCEPTABLE(m). The associated value of this event is k, k'' or sk_s^j.
- **testpw!**(S, j, C, π): SERVER ACTION 2 query to Π_j^S is made with input k', and previously a testpw$(S, j, C, \pi, 3)$ event occurs with associated value k'.
- **testpw**$^*(S, j, C, \pi)$: testpw(S, j, C, π, l) event occurs for some $l \in \{2, 3, 4\}$.
- **testpw**(C, i, S, j, π) : for some $l \in \{2, 3, 4\}$, both a testpw(C, i, S, π, l) and testpw(S, j, C, π, l) event occur, where Π_i^C is *paired* with Π_j^S, and Π_j^S is *paired* with Π_i^C after its SERVER ACTION 1 query.
- **testexecpw**(C, i, S, j, π): for some m, μ and γ', $\mathcal{A}$ makes an $H_l(C, S, m,\ \mu,$ $\sigma,\ \gamma')$ query, for $l \in \{2, 3, 4\}$, and previously made i) an Execute(C, i, S, j)

query that generates m, μ, and ii) an $H_1(\pi)$ query returning $(\gamma')^{-1}$, where $\sigma = DH(\alpha, \mu)$ and $m = \alpha \cdot (\gamma')^{-1}$.

- **correctpw**: before any Corrupt query, either a testpw!(C, i, S, π_C) event occurs for some C,i and S, or a testpw$^*(S, j, C, \pi_C)$ event occurs for some S, j, and C.
- **doublepwserver**: before any Corrupt query, both testpw$^*(S, j, C, \pi)$ event and a testpw$^*(S, j, C, \hat{\pi})$, for some S, j, C and $\pi \neq \hat{\pi}$.
- **pairedpwguess**: a testpw(C, i, S, j, π_C) event occurs, for some C, i, S and j

B Hash function simulation

***H*₁**: For each hash query $H_1(\pi)$, if the same query was previously asked, the simulator retrieves the record (π, Φ, ψ_1) from the list $\mathcal{L}_{h1}$ and answers with Φ. Otherwise, the answer Φ is chosen according to the following rule:

$\star$ **Rule** H_1
Choose $\psi_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$. Compute $\Phi := g^{\psi_1}$ and write the record (π, Φ, ψ_1) to $\mathcal{L}_{h1}$.

Fig. 3: Simulation of the hash function H_1