An extended abstract of this work appears in B. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, Lecture Notes in Computer Science Vol. ????, pages ???–???, Chennai, India, Dec. 4–8, 2005. Springer-Verlag, Berlin, Germany.

A Simple Threshold Authenticated Key Exchange from Short Secrets

Michel Abdalla[1], Olivier Chevassut[2], Pierre-Alain Fouque[1], and David Pointcheval[1]

[1] Departement d'Informatique
École normale supérieure
45 Rue d'Ulm, 75230 Paris Cedex 05, France
{Michel.Abdalla,Pierre-Alain.Fouque,David.Pointcheval}@ens.fr
http://www.di.ens.fr/~{mabdalla,fouque,pointche}
[2] Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA
OChevassut@lbl.gov
http://www.dsd.lbl.gov/~chevassu

Abstract. This paper brings the password-based authenticated key exchange (PAKE) problem closer to practice. It takes into account the presence of firewalls when clients communicate with authentication servers. An authentication server can indeed be seen as two distinct entities, namely a gateway (which is the direct interlocutor of the client) and a back-end server (which is the only one able to check the identity of the client). The goal in this setting is to achieve both transparency and security for the client. And to achieve these goals, the most appropriate choices seem to be to keep the client's password private —even from the back-end server— and to use threshold-based cryptography. In this paper, we present the Threshold Password-based Authenticated Key Exchange (GTPAKE) system: GTPAKE uses a pair of public/private keys and, unlike traditional threshold-based constructions, shares only the private key among the servers. The system does no require any certification —except during the registration and update of clients' passwords— since clients do not use the public-key to authenticate to the gateway. Clients only need to have their password in hand. In addition to client security, this paper also presents highly-desirable security properties such as *server password protection* against dishonest gateways and *key privacy* against curious authentication servers.

Keywords. Threshold Protocols, Password-based Authentication.

1 Introduction

Problem Description. Consider the familiar scenario where you are at the airport waiting for your flight. You have checked-in and have now half an hour to kill. What do you do? Turn on your laptop, switch on your wireless card, and pick up the airport wireless LAN. You are prompted for a password to authenticate yourself and upon successful authentication a port is opened for you to browse the Internet and/or read your e-mails. Now you may wonder what happens under the hood. We have indeed talked to an airport gateway, often termed hotspot, that has in turn talked to your mobile-Internet provider. T-Mobile is an example of such a provider in the United States. The gateway has passed —in an encrypted form— your password to the provider for authentication and gets back a yes/no depending whether or not the authentication was successful.

Although this model is very attractive in practice, existing security solutions implementing it have major drawbacks since the gateway gains some amount of information about your password. The ideal solution is a cryptographic algorithm allowing the client to securely exchange a session key with the gateway, but the gateway does not gain any information about the password and the authentication server does not gain any information about the session key. Additional problems also occur if too many people, from the same provider, try to connect to various gateways at the same time. The authentication check from the provider would become a bottleneck. Various authentication points are very desirable. Nevertheless, the password of the client cannot be stored at several places, otherwise the job of hackers would be made much easier.

Scenario. To formally define a model for the above scenario, we propose a model in which one can design a protocol among three parties (the client, the gateway and the authentication server) which protects both the session keys and the passwords. We indeed require the three following security notions which capture dishonest behaviors of the client, the authentication server and the gateway respectively: the *semantic security* of the sessions keys, which we model by a Real-Or-Random (ROR) game [1] (it has been proved strictly stronger than the more classical Find-Then-Guess (FTG) model [3]); the *key-privacy* notion [1] which entails that the session key exchanged by two parties with the help of an authentication server is unknown to the authentication server (and also to any other party, granted the semantic security); the *server password protection* which basically means that the gateway cannot learn any information about clients' passwords from the authentication server.

The ultimate goal of this paper is to achieve the highest level of security in the PAKE setting. With the above security notions, breaking into the gateway would not help to gain any information about the passwords, however the authentication server is a security hole. Breaking into the latter would leak the authentication information. Furthermore, according to the above motivation example, a unique authentication point may be a bottleneck. When data information is crucial, a usual solution to protect it is to distribute it among several servers such that a majority of them is needed to recover the initial data. Moreover, when we want to protect a cryptographic service we can split the private information into several parts, each known by one server, so that a majority of them is required to maintain the service without reconstructing the secret key in a single place. Threshold cryptography is the field that provides such solutions and allows to take into account adversaries that can break into any minority of parties. It furthermore solves the bottleneck problem.

Contribution. Our contribution in this paper is a provably-secure protocol satisfying the previously mentioned requirements. We have constructed it by defining a simple protocol, called the gateway PAKE (GPAKE) protocol, among three parties (the client, the gateway and the authentication server). GPAKE protects the session keys and the passwords according to a formal security model which we specify in this paper. It provides the additional property to be a variant of AuthA [4] perfectly *transparent* to the client. Transparency means that a client does not (need to) know whether he is talking to a server directly or whether the server is implemented as a gateway, an authentication server, or even an application server. The gateway does not also (need to) know whether the authentication server is distributed. A non-transparent protocol indeed raises real concerns on the utilization of the protocol in practice since clients need to first update their cryptographic stack in order to take advantage of the threshold PAKE feature. A transparent protocol on the other hand lets only domain administrators worry about deploying the threshold PAKE feature to their users. We have developed a threshold version (called gateway threshold PAKE (GTPAKE)) that does not break the transparency property of GPAKE, and defined its execution in our security model. Clients already running the two-party AuthA protocol (e.g, OMDHKE [7]) will not have to upgrade their stack when administrators add an extra layer of protection with GTPAKE!

Related Work. Several password-based key exchange protocols in the threshold setting have been proposed in the past by MacKenzie et al. [13] and by Di Raimondo and Gennaro [9], to name a few. In particular, the protocol by Di Raimondo and Gennaro in Eurocrypt 2003 [9], which is a threshold version of the 2-party KOY protocol [11], is proven secure in the standard model. In that paper, they have introduced the notion of *transparent* protocols, where the initial protocol and its threshold version are the same from the point of view of the client. Unfortunately, their solution is not very efficient from a practical point of view since it requires several rounds of communication between the client and the server and among the servers themselves. Moreover, like previous proposals of threshold password-based key exchange protocols, their protocol requires the password to be shared among all the servers.

The solution we propose in this paper does not require passwords to be shared across different servers. Instead, we only share the secret decryption key for a public-key encryption scheme under which all passwords are encrypted. This provides an additional feature: it is quite easy for a client to modify his password. He just needs to send the new one encrypted under the authentication servers' public key.

Moreover, contrary to the hybrid model of Halevi and Krawczyk [10], where the server has a public/secret key pair and the client only knows a password, the client is not required to check the authenticity of the public key during the execution of the protocol. Integrity is required only during the registration or when the user wants to update his password.

Organization of the Paper. In Section 2, we present the formalization used to define the execution of the GPAKE and GTPAKE protocols. Our formalization extends that of Abdalla et al. to the threshold setting [1]. In Section 3, we present the intractability assumptions used throughout the paper. In Section 4, we describe the GPAKE system and show that it achieves *semantic security* and *key privacy* in a provable secure way. In Section 5, we describe the GPAKE system's threshold version and show that it is secure —via a reduction from the security of GTPAKE to the intractability assumptions— against dictionary attacks.

2 Security Model

In this section, we present the security model we will use in the rest of the paper to define the execution of our protocol for threshold password-authenticated key exchange.

2.1 Overview

Gateway-oriented password-based key exchange. A gateway-oriented password-based key exchange is a three-party protocol among a client, a gateway, and an authentication server. The goal of protocol is to establish an implicitly authenticated session key between the client and the gateway with the help of the authentication server, where the authentication is done by means of a short password. In our model, the password is known to both the client and the authentication server, but not to the gateway. In fact, no long-term secrets are stored in the gateway. The authentication server, on the other hand, is assumed to know the password. While the communication channel between the gateway and the authentication server is assumed to be authenticated and private, the channel connecting the client to the gateway may be insecure and perhaps under the control of an adversary.

The security goals of our gateway-oriented password-based key exchange model are also somewhat different from those of previous models for password-based schemes. In particular, we ask that the session key shared between the gateway and the client should remain private to the authentication server (see Section 2.2 for more details). Moreover, we also ask that the chances of the gateway learning some information on the password after multiple interactions with the server, perhaps concurrently, should be negligible.

Protocol participants. The participants in a gateway-oriented password-based key exchange are the client $C \in \mathcal{C}$, the gateway $G \in \mathcal{G}$, and the authentication server $S \in \mathcal{S}$. We denote by $\mathcal{U}$ the set of all participants (i.e., $\mathcal{U} = \mathcal{C} \cup \mathcal{G} \cup \mathcal{S}$) and by U a non-specific participant in $\mathcal{U}$. Each client $C \in \mathcal{C}$ holds a password pw_C. Each server $S \in \mathcal{S}$ holds a vector of passwords $\mathsf{PW}_S = \langle \mathsf{pw}_C \rangle_{C \in \mathcal{C}}$ with an entry for each client.

2.2 Security Model

Since we assume an authenticated and private channel between the gateway and the server, the communication model is similar to previous one for 2-party authenticated key exchange. In particular, we adopt the Real-Or-Random (ROR) security model of Abdalla *et al.* [1] for password-based authenticated key exchange protocol, which in turn implies that of Bellare *et al.* [3]. As in the standard model, all the interactions between an adversary $\mathcal{A}$ and the protocol participants in the ROR model are done via oracle queries. Let U^i denote the instance i of a participant U. The list of oracles available to the adversary are as follows:

- $\mathsf{Execute}(C^i, G^j)$: This query models passive attacks in which the attacker eavesdrops on honest executions among a client instance C^i and a gateway instance G^k. The output of this query consists of the messages that were exchanged during the honest execution of the protocol.
- $\mathsf{Send}(U^i, m)$: This query models an active attack against the client or gateway instance U^i, in which the adversary may intercept a message and then modify it, create a new one, or simply forward it to the intended recipient. The output of this query is the message that the participant instance U^i would generate upon receipt of message m.

The Real-Or-Random Model [1]. In the ROR model, in addition to the above-mentioned oracles, an attacker is also given access to a less restrictive Test oracle. Let b be a bit chosen uniformly at random at the beginning of the experiment defining the semantic security of session keys. The Test oracle in the ROR model is defined as follows:

- $\mathsf{Test}(U^i)$: If no session key for instance U^i is defined, then return the undefined symbol $\perp$. Otherwise, return the session key for instance U^i if $b = 1$ or a random of key of the same size if $b = 0$.

As in standard models, the Test oracle in the ROR model also tries to capture the adversary's ability (or inability) to tell apart a real session key from a random one. The main difference is that it does so not only for a single session but for all sessions. More precisely, the adversary in the ROR model is not restricted to ask a single Test query, but it can in fact ask multiple ones. All Test queries in this case will be answered using the same value for the hidden bit b that was chosen at the beginning of the experiment defining the semantic security of the session keys. That is, the keys returned by the Test oracle are either all real or all random. However, in the random case, the same random key value is returned for Test queries that are asked to two instances that belong to the same session (see notion of partnering below). The goal of the adversary in the ROR model is still the same: to guess the value of the hidden bit b used to answer Test queries. The adversary is considered successful if it guesses b correctly.

Partnering. As in [1], we use the notion of partnering based on session identifications (*sid*), which says that two instances are partnered if they hold the same non-null *sid*. More specifically, a client instance C^i and a gateway instance G^j are said to be partners if the following conditions are met: (1) Both C^i and G^j accept; (2) Both C^i and G^j share the same session identifications; (3) The partner identification for C^i is G^j and vice-versa; and (4) No instance other than C^i and G^j accepts with a partner identification equal to C^i or G^j. In practice, the *sid* is taken to be the partial transcript of the conversation among the client and the gateway instances before the acceptance.

FRESHNESS. Differently from [1], we opt not to embed the notion of freshness inside the definition of the oracles. Instead, we take the more standard approach of explicitly defining the notion of freshness and mandating the adversary to only perform tests on *fresh* instances. The two approaches are however equivalent. In particular, we say that a *instance* of a client or gateway is *fresh* if it has accepted.

FORMAL DEFINITION. Let SUCC denote the event in which the adversary is successful. The $\mathsf{ake}-\mathsf{ror}$**-advantage** of an adversary $\mathcal{A}$ in violating the semantic security of the protocol P in the ROR sense and the **advantage function** of the protocol P, when passwords are drawn from a dictionary Dict, are respectively

$$\mathbf{Adv}^{\mathrm{ake-ror}}_{P,\mathsf{Dict}}(\mathcal{A}) = 2 \cdot \Pr[\,\textsc{Succ}\,] - 1 \text{ and}$$
$$\mathbf{Adv}^{\mathrm{ake-ror}}_{P,\mathsf{Dict}}(t, R) = \max_{\mathcal{A}}\{\, \mathbf{Adv}^{\mathrm{ake-ror}}_{P,\mathsf{Dict}}(\mathcal{A})\,\},$$

where the maximum is over all $\mathcal{A}$ with time-complexity at most t and using resources at most R (such as the number of queries to its oracles). The definition of time-complexity that we use henceforth is the usual one, which includes the maximum of all execution times in the experiments defining the security plus the code size.

Please note that, as proven in [1], any scheme proven secure in the ROR model is also secure in the model of Bellare *et al.* [3]. The converse, however, is not necessarily true (see [1] for more details).

AUTHENTICATION. The notion of semantic security does not guarantee the existence of a partner, but only the secrecy of the session key (implicit authentication). In order to address this problem, one usually adds mechanisms for explicit authentication of client and gateway instances. In this paper, we only consider *unilateral authentication* of the gateway, by which a client instance can be ensured that it has in fact established a key with the gateway instance it intended to. As in [6], we denote by $\mathsf{Succ}^{\mathrm{G-auth}}_{\mathcal{A}}$ the probability that adversary $\mathcal{A}$ successfully impersonates the gateway in an execution of the protocol. This is the probability with which a client instance accepts without having a gateway partner. The advantage function of the protocol can be defined as in previous cases.

Key Privacy. The notion of key privacy was introduced in [1] to capture the idea that the session key computed by two parties with the aid of an authentication server should only be known to those two parties and not to the server. The goal in this case is to reduce the amount of trust one puts into the server. In order to meet this goal, one has to consider an adversary with access to all the secret information stored in the server and then show that such adversary cannot distinguish actual session keys from random ones if we restrict this adversary to test sessions in which the keys are shared between two oracles. The latter restriction is important since an adversary with access to the secrets of the authentication server could always establish a key with a client by playing the roles of the gateway and authentication server. Since one of our main goals is to show that the key shared between the client and the gateway is not known to the authentication server, we also use the notion of key privacy.

To capture the above intuition more formally, [1] considers an adversary which has access to all the secrets held by the authentication server and to the oracles used in the experiment defining semantic security. They then introduce a new type of oracle, called TestPair, whose goal is to capture the adversary's ability to distinguish the real session key shared between any two oracle instances from a random one. The inputs to the TestPair oracle are the specific oracle instances whose shared session key the adversary thinks it can tell apart from a random key.

FORMAL DEFINITION. Consider an execution of the key exchange protocol P by an adversary $\mathcal{A}$ with access to all the secret held by the authentication server as well as to the Execute, Send, and TestPair oracles. Let SUCC denote the event in which the adversary is successful in guessing the hidden bit used by TestPair oracle when only asking TestPair queries to instances pairs that have accepted. The **kp-advantage** of an adversary $\mathcal{A}$ in violating the key privacy of the protocol P in the ROR sense ($\mathbf{Adv}^{\mathrm{ake-kp}}_{P,\mathsf{Dict}}(\mathcal{A})$) and the **advantage function** of P ($\mathbf{Adv}^{\mathrm{ake-kp}}_{P,\mathsf{Dict}}(t, R)$), when passwords are drawn from a dictionary Dict, are then defined as in previous definitions.

Server Password Protection. As we mentioned earlier, one of the goals of our protocol is to guarantee that the gateway is not capable of learning the user's password that is stored in the server. Clearly, as in the case of semantic security, one cannot hope for much since, in each interaction, the adversary may be able to eliminate one candidate password from the list of possible passwords. However, we ask that the adversary should not be able to do much better than. That is, if the adversary interacts q times with the server, then the probability that it can distinguish the true password from a random one in the dictionary should be only negligibly larger than $O(q/N)$, where N is the size of the dictionary. The hidden constant in this case should be as small as possible (preferably 1). Note that, in this definition, the dictionary is assumed to be uniformly distributed.

2.3 Threshold Security Model

THRESHOLD GATEWAY-ORIENTED PASSWORD-BASED KEY EXCHANGE. A (t, k, n)-threshold gateway-oriented password-based key exchange is an extension of the basic gateway-oriented password-based key exchange in which the authentication server is a distributed entity. More specifically, the clients' passwords are no longer known to any single server. Instead, each server in the set of n authentication servers is assumed to hold a share of the secret key of a public-key encryption scheme, under which clients' passwords are encrypted. The authentication of any client will require the cooperation of some size-k subset of honest servers. In addition, any adversary who learns t or fewer shares of the secret key should not learn any information about the clients' passwords.

PARTICIPANTS. The participants in a threshold protocol are the client $C \in \mathcal{C}$, the gateway $G \in \mathcal{G}$, the set of authentication servers $\{S_1, \ldots, S_n\}$ with $S_i \in \mathcal{S}$, and a trusted dealer.

Semantic Security. The definition of semantic security of threshold protocols follows the one given above for gateway-oriented protocols. At the beginning of the semantic security experiment, the adversary selects a subset of at most $t = k - 1$ servers to corrupt. We say that the adversary is *static* when it chooses the set of servers to corrupt in advance, before seeing anything. A special server, called the combiner, will be used to perform some tasks that do not require any secret. The combiner is also in charge of all communications between the gateway and the other servers.

The dealer generates a public key pk and a secret key sk for an encryption scheme. Then, he performs the secret sharing of sk and sends the part sk_i to P_i along with a verification key vk_i. The adversary obtains the secret key shares of the corrupted servers, along with the public key and the verification keys.

After this phase, the adversary is given access to the same set of oracles used in the standard security model for gateway-oriented password-based authenticated key exchange protocols.

Robustness. We say that a threshold scheme is *robust* when it takes into account *malicious* adversaries whose behavior can be different from the protocol. To force the servers to correctly perform their job, we use proofs of validity in our protocol. This also enables the combiner to correctly decrypt.

3 Diffie-Hellman Assumptions

In this section, we recall the definitions of standard Diffie-Hellman assumptions and introduce some new variants, which we use in the security proof of our protocol. We also present some relations between these assumptions.

3.1 Classical Assumptions

Henceforth, we assume a finite cyclic group G of prime order p generated by an element g. We also call the tuple $\mathbb{G} = (G, g, p)$ a represented group.

Computational Diffie-Hellman Assumption: CDH. The CDH assumption, in a represented group $\mathbb{G}$, with respect to the basis X, states that given two elements $X' = X^u$ and $Y = X^v$, where u and v were drawn at random from Z_p, it is hard to compute $Y' = Y^u = X^{uv}$. This can be defined more precisely by considering an experiment $\mathbf{Exp}_{\mathbb{G}}^{\mathrm{cdh}}(\mathcal{A}, X)$, in which we select an exponent u in Z_p, an element Y in G, compute $X' = X^u$, and then give both X' and Y to $\mathcal{A}$. Let Y' be the output of $\mathcal{A}$. Then, the experiment $\mathbf{Exp}_{\mathbb{G}}^{\mathrm{cdh}}(\mathcal{A}, X)$ outputs 1 if $Y' = Y^u$ and 0 otherwise. We define the *advantage* of $\mathcal{A}$ in violating the CDH assumption with respect to X as $\mathbf{Adv}_{\mathbb{G}}^{\mathrm{cdh}}(\mathcal{A}, X) = \Pr[\,\mathbf{Exp}_{\mathbb{G}}^{\mathrm{cdh}}(\mathcal{A}, X) = 1\,]$, the *advantage* of $\mathcal{A}$ in violating the CDH assumption (with a random basis) as $\mathbf{Adv}_{\mathbb{G}}^{\mathrm{cdh}}(\mathcal{A}) = \mathbf{E}_X\left[\mathbf{Adv}_{\mathbb{G}}^{\mathrm{cdh}}(\mathcal{A}, X)\right]$, and the *advantage functions*, $\mathbf{Adv}_{\mathbb{G}}^{\mathrm{cdh}}(t, X)$ and $\mathbf{Adv}_{\mathbb{G}}^{\mathrm{cdh}}(t)$, as the maximum values of $\mathbf{Adv}_{\mathbb{G}}^{\mathrm{cdh}}(\mathcal{A}, X)$ and $\mathbf{Adv}_{\mathbb{G}}^{\mathrm{cdh}}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t.

It is also often assumed that, *independently of what the value of X is (as long as it is a generator, of order p)*, the CDH problem with respect to the basis X is hard: the maximal value of $\mathbf{Adv}_{\mathbb{G}}^{\mathrm{cdh}}(t, X)$ over all generators X is small for any reasonable t.

Decisional Diffie-Hellman Assumption: DDH. Roughly, the DDH assumption, with respect to the basis X, states that the distributions $(X, X' = X^u, Y, Y' = Y^u)$ and $(X, X' = X^u, Y, Z = Y^v)$ are computationally indistinguishable when Y is drawn at random from G, and u and v are drawn at random from Z_p. As before, we can define the DDH assumption more formally by defining two experiments, $\mathbf{Exp}_{\mathbb{G}}^{\mathrm{ddh\text{-}real}}(\mathcal{A}, X)$ and $\mathbf{Exp}_{\mathbb{G}}^{\mathrm{ddh\text{-}rand}}(\mathcal{A}, X)$. In both experiments, we compute two random values $X' = X^u$ and Y as before. But in addition to that, we also provide a third input, which is Y^u in $\mathbf{Exp}_{\mathbb{G}}^{\mathrm{ddh\text{-}real}}(\mathcal{A}, X)$ and Y^v, for a random v, in $\mathbf{Exp}_{\mathbb{G}}^{\mathrm{ddh\text{-}rand}}(\mathcal{A}, X)$. The goal of the adversary is to guess a bit indicating the experiment he thinks he is in. We define the *advantage* of $\mathcal{A}$ in violating the DDH assumption, with respect to the basis X, $\mathbf{Adv}_{\mathbb{G}}^{\mathrm{ddh}}(\mathcal{A}, X)$, as

$\Pr[\,\mathbf{Exp}^{\text{ddh-real}}_{\mathbb{G}}(\mathcal{A}, X) = 1\,] - \Pr[\,\mathbf{Exp}^{\text{ddh-rand}}_{\mathbb{G}}(\mathcal{A}, X) = 1\,]$, and *advantage* of $\mathcal{A}$ in violating the DDH assumption (with random basis) as $\mathbf{Adv}^{\text{ddh}}_{\mathbb{G}}(\mathcal{A}) = \mathbf{E}_X\left[\mathbf{Adv}^{\text{ddh}}_{\mathbb{G}}(\mathcal{A}, X)\right]$. The *advantage functions* $\mathbf{Adv}^{\text{ddh}}_{\mathbb{G}}(t, X)$ and $\mathbf{Adv}^{\text{ddh}}_{\mathbb{G}}(t)$ are then defined in a similar manner as above.

Again, it is also often assumed that, *independently of what X is (as long as it is a generator, of order p)*, the DDH problem with respect to the basis X is hard: the maximal value of $\mathbf{Adv}^{\text{ddh}}_{\mathbb{G}}(t, X)$ over all generators X is small for any reasonable t.

3.2 Password-Based Chosen-Basis Diffie-Hellman Assumptions

The actual proofs of security of our protocol use password-related versions of the above Diffie-Hellman assumptions, in which the adversary has some control over the basis, hence the name *password-based chosen-basis decisional/computational Diffie-Hellman* assumptions. They make use of a dictionary $\mathcal{D} = \{U_1, \ldots, U_N\}$ of size N. Then, we assume that when the adversary has not correctly predicted the password (1 chance over N), he has no significant advantage. Hence, its overall advantage cannot be significantly larger than $1/N$.

We start by presenting the password-based chosen-basis computational Diffie-Hellman assumption.

Definition 1 (PCCDH**).** Let $\mathbb{G} = (G, g, p)$ be a represented group and let $\mathcal{A}$ be an adversary. Consider the following experiment, where $\mathcal{D}$ is a dictionary of N elements in G.

$$
\begin{array}{l}
\textbf{Experiment } \mathbf{Exp}^{\text{pccdh}}_{\mathbb{G}}(\mathcal{A}, \mathcal{D}) \\
\quad (X, s) \leftarrow \mathcal{A}(\mathsf{find}, \mathcal{D}) \\
\quad \Pi \stackrel{R}{\leftarrow} \mathcal{D}\ ;\ Y \stackrel{R}{\leftarrow} G \\
\quad K \leftarrow \mathcal{A}(\mathsf{guess}, s, Y, \Pi) \\
\quad \textbf{return } \ 1 \text{ if } K = \mathrm{CDH}(X/\Pi, Y)
\end{array}
$$

We define the *advantage* of $\mathcal{A}$ in violating the PCCDH assumption with respect to the dictionary $\mathcal{D}$, $\mathbf{Adv}^{\text{pccdh}}_{\mathbb{G},N}(\mathcal{A}, \mathcal{D})$, the *advantage* of $\mathcal{A}$ $\mathbf{Adv}^{\text{pccdh}}_{\mathbb{G},N}(\mathcal{A})$, and the *advantage functions*, $\mathbf{Adv}^{\text{pccdh}}_{\mathbb{G},N}(t, \mathcal{D})$ and $\mathbf{Adv}^{\text{pccdh}}_{\mathbb{G},N}(t)$, as above. $\diamondsuit$

In our security proofs, we actually need a simple variation of the above problem, in which the adversary, in the second stage, outputs a set of s candidates for the CDH value. The adversary wins if the set indeed contains K. This problem is thus named *Set Password-based Chosen-basis Computational Diffie-Hellman* Problem (SPCCDH).

We define the *advantage* $\mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N}(\mathcal{A}, \mathcal{D}, s)$ of $\mathcal{A}$ in violating the SPCCDH assumption with respect to the dictionary $\mathcal{D}$, the *advantage* $\mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N}(\mathcal{A}, s)$ of $\mathcal{A}$, and the *advantage functions* $\mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N}(t, \mathcal{D}, s)$ and $\mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N}(t, s)$ as in previous definitions.

Fortunately, the two new assumptions are not so strong, since one can prove that the SPCCDH problem is equivalent to the CDH problem as proven in Appendix C. The general result proven in Appendix C can be simplified in the particular case of not so large dictionaries:

Lemma 2. $\mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N}(t, s) \leq \frac{1}{N} + N^2 s^2 \times \mathbf{Adv}^{\text{cdh}}_{\mathbb{G}}(2t + \tau_e)$.

In addition to the computational assumptions above, we also make use of the following decisional assumption in our security proofs.

Definition 3 (PCDDH**).** Let $\mathbb{G} = (G, g, p)$ be a represented group and let $\mathcal{A}$ be an adversary. Consider the following experiment, defined for $b = 0, 1$, where $\mathcal{D}$ is the dictionary of N elements in G.

$$
\begin{array}{l}
\textbf{Experiment } \ \mathbf{Exp}_b^{\mathrm{pcddh}}(\mathcal{A},\mathcal{D}) \\
\quad (X,Y,s) \leftarrow \mathcal{A}(\mathsf{find},\mathcal{D}) \\
\quad \Pi \stackrel{R}{\leftarrow} \mathcal{D}\ ;\ \ s_0, s_1 \stackrel{R}{\leftarrow} \mathsf{Z}_p \\
\quad Y' \leftarrow Y^{s_0}\ ;\ \ X' \leftarrow (X/\Pi)^{s_b} \\
\quad b' \leftarrow \mathcal{A}(\mathsf{guess}, s, X', Y', \Pi) \\
\quad \textbf{return } \ b'
\end{array}
$$

We define the *advantage* of $\mathcal{A}$ in violating the PCDDH assumption with respect to the dictionary $\mathcal{D}$, $\mathbf{Adv}_{\mathbb{G},N}^{\mathrm{pcddh}}(\mathcal{A},\mathcal{D})$, the *advantage* of $\mathcal{A}$, $\mathbf{Adv}_{\mathbb{G},N}^{\mathrm{pcddh}}(\mathcal{A})$, and the respective *advantage functions* of $\mathbb{G}$ for a given value N, $\mathbf{Adv}_{\mathbb{G},N}^{\mathrm{pcddh}}(t,\mathcal{D})$ and $\mathbf{Adv}_{\mathbb{G},N}^{\mathrm{pcddh}}(t)$, as above. $\diamondsuit$

Fortunately again, this problem is not new. It has already appeared in [2] under the name PCDDH2. In that paper, the authors have also shown that $\mathbf{Adv}_{\mathbb{G},N}^{\mathrm{pcddh}}(t,\mathcal{D})$ and $\mathbf{Adv}_{\mathbb{G},N}^{\mathrm{pcddh}}(t)$ cannot be significantly larger than $2/N$.

4 The Gateway PAKE System

In this section, we describe GPAKE, the underlying gateway-oriented password-based protocol used in the construction of our threshold gateway-oriented password-based protocol.

4.1 Description

Our gateway-oriented password-based protocol, GPAKE, builds upon previous password-based key exchange protocols in [4, 7, 12], which in turn are based on the encrypted key exchange of Bellovin and Merritt [5]. Its description is given in Figure 1, where $\mathbb{G} = (G, g, q)$ is a represented group; ℓ is a security parameter; and $\mathcal{G} : \mathcal{U}^2 \times \mathsf{Dict} \rightarrow \mathbb{G}$, $\mathsf{Hash}_1 : \mathcal{U}^2 \times \mathbb{G} \times \mathbb{G} \rightarrow \{0,1\}^\ell$, and $\mathsf{Hash}_2 : \mathcal{U}^2 \times \mathbb{G} \times \mathbb{G} \rightarrow \{0,1\}^\ell$, are random oracles.

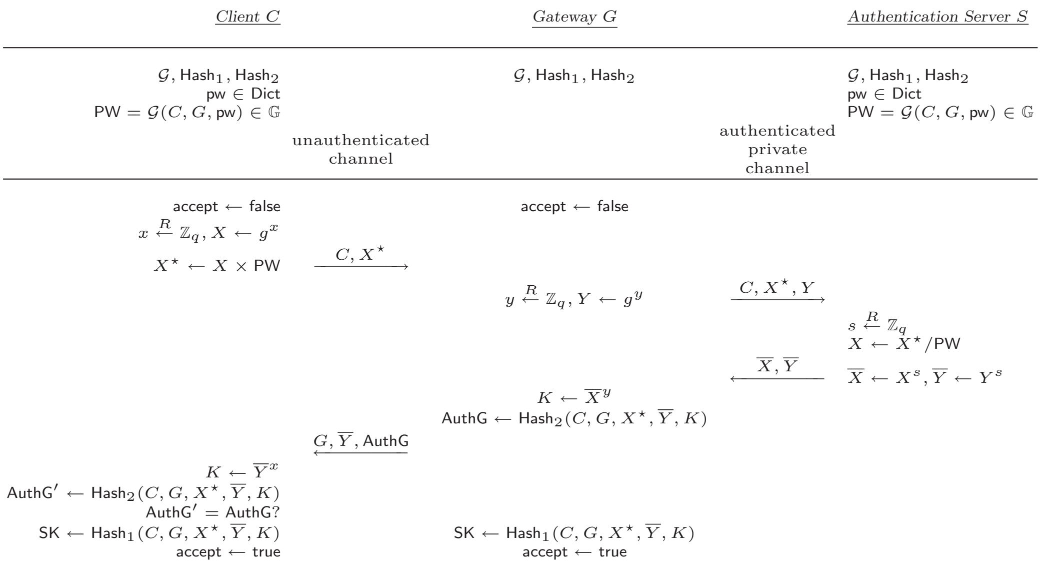

Fig. 1. GPAKE: A gateway-oriented password-based authenticated key exchange protocol.

The protocol consists of four message exchanges, two between the client and the gateway and two between the gateway and the authentication server. Since the channel connecting the gateway to the server is assumed to be authenticated and private, from the client perspective, the protocol resembles almost exactly the 2-party protocol OMDHKE in [7]. The only difference is in the key derivation function, which does not include the password.

The protocol starts with the client choosing a random element $x \in \mathsf{Z}_p$ and computing $X = g^x$, and encrypting it using $\mathcal{G}(C, G, \mathsf{pw})$ as a mask to obtain $X^\star$. The client then sends to the gateway both $X^\star$ and its identity string C. Upon receiving a message from the client, the gateway chooses a random element $y \in \mathsf{Z}_p$ and computes $Y = g^y$, and forwards to the server both Y and the value $X^\star$ that it has received from the client. Upon receiving the values $(X^\star, Y)$ from the gateway, the server computes $X = X^\star / \mathcal{G}(C, G, \mathsf{pw})$, chooses a random element $s \in \mathsf{Z}_p$, computes $(\overline{X} = X^s, \overline{Y} = Y^s)$, and sends it back to the gateway. Upon receiving $(\overline{X}, \overline{Y})$, the gateway computes the Diffie-Hellman key $K = \overline{X}^y$, the authenticator $\mathsf{AuthG} = \mathsf{Hash}_2(C, G, X^\star, \overline{Y}, K)$ and the session key $\mathsf{SK} = \mathsf{Hash}_1(C, G, X^\star, \overline{Y}, K)$, and sends $(G, \overline{Y}, \mathsf{AuthG})$ to the client. Upon receiving $(G, \overline{Y}, \mathsf{AuthG})$, the client computes the Diffie-Hellman key $K = \overline{Y}^x$, checks whether $\mathsf{AuthG} = \mathsf{Hash}_2(C, G, X^\star, \overline{Y}, K)$, and sets the session key to $\mathsf{SK} = \mathsf{Hash}_1(C, G, X^\star, \overline{Y}, K)$ if the test passes. The session identification is defined to be the transcript $(C, G, X^\star, \overline{Y})$ of the conversation between the client and the gateway.

4.2 Security

Semantic security. As the following theorem states, GPAKE is a secure gateway-oriented password-based key exchange protocol as long as the SPCCDH problem is hard in $\mathbb{G}$. As shown in Section 3, this is equivalent to assuming that CDH problem is hard in $\mathbb{G}$. The proof can be found in Appendix B. Nevertheless, we note here that the proof of security assumes Dict to be a uniformly distributed dictionary and of size smaller than 2^ℓ.

Theorem 4. *Let $\mathbb{G} = (G, g, q)$ be a represent group of prime order q and let Dict be a uniformly distributed dictionary of size $N = |\mathsf{Dict}|$. Let GPAKE describe the gateway-oriented protocol associated with these primitives as defined in Figure 1. Then,*

$$\mathbf{Adv}^{\mathrm{ake-ror}}_{\mathsf{GPAKE},\mathbb{G},\mathsf{Dict}}(t, q_{\mathrm{exe}}, q_{\mathrm{fake-C}}, q_{\mathrm{fake-G}}, q_{\mathrm{active}}, q_{\mathrm{test}}, q_{\mathrm{Hash}_1}, q_{\mathrm{Hash}_2}, q_{\mathcal{G}}) \leq$$

$$\begin{aligned}
&\frac{q^2_{\mathrm{active}} + q^2_{\mathcal{G}}}{q} + \frac{q^2_{\mathrm{exe}}}{q^2} + 2\,\frac{q^2_{\mathrm{Hash}_1} + q^2_{\mathrm{Hash}_2}}{2^\ell} + \\
&2\,(q_{\mathrm{Hash}_1} + q_{\mathrm{Hash}_2}) \cdot \mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(t + (4q_{\mathrm{exe}} + 4)\tau_e) + \\
&2 \cdot q_{\mathrm{active}} \cdot (q_{\mathrm{Hash}_1} + q_{\mathrm{Hash}_2}) \cdot \mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(t + 2\tau_e) + 4 \cdot q_{\mathrm{fake-C}}/N + \\
&2 \cdot q_{\mathrm{fake-G}} \cdot \mathbf{Adv}^{\mathrm{spccdh}}_{\mathbb{G},N}(t, q_{\mathrm{Hash}_1} + q_{\mathrm{Hash}_2})\,,
\end{aligned}$$

where $q_{\mathcal{G}}$, q_{Hash_1}, and q_{Hash_2} represent the number of queries to the $\mathcal{G}$, Hash_1 and Hash_2 oracles, respectively; q_{exe} represents the number of queries to the $\mathsf{Execute}$ oracle; $q_{\mathrm{fake-C}}$ and $q_{\mathrm{fake-G}}$ represent the number of attempts of the adversary to fake the client and the gateway, respectively; q_{active} represents the total number of queries to the Send oracle; q_{test} represents the total number of queries to the Test oracle; and τ_e denotes the exponentiation computational time in $\mathbb{G}$.

Remark 5. In the security model presented in Section 2, the adversary is not allowed to corrupt gateway instances. Consequently, the proof of Theorem 4 does not guarantee the security of GPAKE in that scenario. Even though GPAKE appears to be secure in the presence of such adversaries, a new proof of security would be required in this case.

Key Privacy. As the following shows, GPAKE achieves the goal of key privacy as long as the DDH problem is hard in $\mathbb{G}$.

Theorem 6. *Let $\mathbb{G} = (G, g, q)$ be a represent group of prime order q and let Dict be a uniformly distributed dictionary of size $N = |\mathsf{Dict}|$. Let GPAKE describe the gateway-oriented protocol associated with these primitives as defined in Figure 1. Then,*

$$\mathbf{Adv}^{\mathrm{ake-kp}}_{\mathsf{GPAKE},\mathbb{G},\mathsf{Dict}}(t, q_{\mathrm{exe}}, q_{\mathrm{test}}) \ \leq\ 2 \cdot \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(t + (4q_{\mathrm{exe}} + 4)\tau_e)\ ,$$

where q_{exe} and q_{test} represent the total number of queries to the $\mathsf{Execute}$ and $\mathsf{TestPair}$ oracles; and τ_e denotes the exponentiation computational time in $\mathbb{G}$.

The proof of Theorem 6 is in Appendix B. It is worth mentioning that, when proving the security of GPAKE, we do not give the server access to a Send oracle since we assume the server to be honest but curious. We do so because, in the actual implementation of GPAKE, the server is distributed and we assume the majority of them to be honest (see Section 5). We also note that, in order to prove key privacy in scenarios where the majority of servers is corrupted, additional modifications would need to be made to GPAKE. These modifications would include the addition of an authenticated Diffie-Hellman protocol between C and G as done in [1] and a proof that the pair $(\overline{X}, \overline{Y})$ is well formed.

Server Password Protection. The server password protection of GPAKE follows directly from the password-based chosen-basis decisional Diffie-Hellman assumption (PCDDH) introduced in [2] and recalled in Section 3. More specifically, it is easy to see that the interaction between the gateway and the server corresponds exactly to the security experiment for PCDDH. Since the security of the latter was shown in [2] to be only negligibly larger than $2/N$, where N is the size of the dictionary, it follows (via a standard hybrid argument as in [2]) that, in each interaction with the server, an adversarial gateway cannot do much better than eliminating two passwords from the list of possible candidates with each interaction. As a result, after q interactions with the server, the advantage of a malicious gateway would be only negligibly larger than $2q/N$.

5 The Gateway Threshold PAKE System

In this section, our goal is to distribute the authentication server in the previous gateway-oriented password-based protocol to prevent malicious adversaries that can corrupt up to k out of n servers. The solution is robust against static adversaries. The threshold version is *transparent* from the point of view of the client since it communicates only with the gateway. The threshold version is also *transparent* from the point of view of the gateway since a special authentication server, called the *combiner*, is the only server with which the gateway communicates. We also assume that the channel between the gateway and the combiner is authenticated and private. We can use signature and encryption schemes in order to fulfill this requirement using threshold signature and encryption.

Description. Let $\mathbb{G}$ is a cyclic subgroup of prime order q. We assume that the authentication servers $\{S_1, \ldots, S_n\}$ share a secret ElGamal encryption key $\mathsf{sk} = x$ using Shamir scheme with threshold k. Server i knows $\mathsf{sk}_i = x_i$ and has a verification key $\mathsf{vk}_i = v^{x_i}$, where v is a generator of $\mathbb{G}$. The encryption of the password $c = \mathbf{E}_{\mathsf{pk}}(\mathsf{PW}) = (e_c, f_c)$ under the public ElGamal encryption key $\mathsf{pk} = (g, y)$ is authenticated in a public file. At the beginning of the protocol, the combiner, S_1 wlog, receives $X^\star = X \times \mathsf{PW}$, and $Y = g^y$.

First Stage The combiner encrypts $X^\star$ into (e, f) and proves the validity of the encryption using a zero-knowledge proof of equality of discrete-log $\mathsf{EDLog}_{y,g}(e/X^\star, f)$. Given $X^\star$ and (e, f), where $e = X^\star \times y^r$ and $f = g^r$, then we have to show that $\log_y(e/X^\star) = \log_g(f)$. Then, he broadcasts to all servers $Y, X^\star, (e, f)$ and the proof of validity $\mathsf{EDLog}_{y,g}(e/X^\star, f)$.

Second Stage Each authentication server S_i checks the proofs and chooses a random value $s_i \in \mathbb{Z}_q$. Then, he computes $Y_i = Y^{s_i}$, $e'_i = (e/e_c)^{s_i}$ and $f'_i = (f/f_c)^{s_i}$ along with a proof of validity that Y_i and the two parts of (e'_i, f'_i) have been raised to the same power s_i: $\mathsf{EDLog}_{Y,e/e_c}(Y_i, e'_i)$ and $\mathsf{EDLog}_{Y,f/f_c}(Y_i, f'_i)$. All these values are broadcast to all the servers.

Third Stage They check all proofs and they compute the values $e'^s = \prod_i e'_i = e'^{\sum_i s_i}$, $f'^s = \prod_i f'_i = f'^{\sum_i s_i}$, and $Y^s = \prod_i \tilde{Y}_i$. Then, they perform a threshold decryption of (e'^s, f'^s) by computing $g_i^s = (f'^s)^{x_i}$. Next, they prove the validity of the decryption by computing $\mathsf{EDLog}_{f'^s,v}(g_i^s, \mathsf{vk}_i)$. Finally, they broadcast the proof along with their decryption share g_i^s.

Forth Stage The combiner, without any secret, can compute $f'^{xs} = \prod_{i \in S} g_i^{s\lambda_{0,i}^S}$ by using Lagrange interpolation formula and $\overline{X} = X^s = e'^s/f'^{xs}$ using k valid decryption shares. He sends to the gateway $\overline{X}$ and $\overline{Y} = Y^s$ using the authentication and private channel.

	Gateway G		*Authentication Servers S_i*
unauthenticated channel	$\mathcal{G}, \mathsf{Hash}_1, \mathsf{Hash}_2$	authenticated private channel	$\mathcal{G}, \mathsf{Hash}_1, \mathsf{Hash}_2$ $\mathsf{pw} \in \mathsf{Dict}$ $\mathsf{PW} = \mathcal{G}(C, G, \mathsf{pw}) \in \mathbb{G}$ g, v generators of $\mathbb{G}$, $\mathsf{pk} = y = g^x$, $\mathsf{sk} = x \in \mathbb{Z}_q$ $\mathsf{sk}_i = x_i \in \mathbb{Z}_q$, $\mathsf{vk} = v^{x_i}$ $\mathbf{E}$ ElGamal encryption scheme $c = \mathbf{E}_{\mathsf{pk}}(\mathsf{PW}) = (e_c, f_c)$, $e_c = \mathsf{PW} \times y^k$, $f_c = g^k$
	$\mathsf{accept} \leftarrow \mathsf{false}$		
$\xrightarrow{A, X^\star}$			
	$y \xleftarrow{R} \mathbb{Z}_q, Y \leftarrow g^y$	$\xrightarrow{A, X^\star, Y}$	
			$*S_1$ computes and broadcasts $Y, X^\star, (e, f) = \mathbf{E}_{\mathsf{pk}}(X^\star), \mathsf{EDLog}_{y,g}(e/X^\star, f)$ $*S_i$ checks the proof and computes $s_i \xleftarrow{R} \mathbb{Z}_q, Y_i \leftarrow Y^{s_i}$ and $e'_i \leftarrow (e/e_c)^{s_i}, f'_i \leftarrow (f/f_c)^{s_i}$, and $\mathsf{EDLog}_{Y,e/e_c}(Y_i, e'_i), \mathsf{EDLog}_{Y,f/f_c}(Y_i, f'_i)$ where $(e', f') = \mathbf{E}_{\mathsf{pk}}(X)$ and broadcasts the proofs and Y_i, e'_i, f'_i $*S_i$ checks the proofs and computes $e'^s \leftarrow \prod_i e'_i, f'^s \leftarrow \prod_i f'_i$ and $\overline{Y} \leftarrow Y^s, g_i^s \leftarrow f'^{sx_i}, \mathsf{EDLog}_{f'^s,v}(g_i^s, \mathsf{vk}_i)$ where $s = \sum_i s_i$ if proofs of S_i are correct and broadcasts the proof and g_i^s $*S_1$ checks the proofs and decrypts $\overline{X} \leftarrow e'^s/f'^{xs}$ where $f'^{xs} \leftarrow \prod_{i \in S} g_i^{s\lambda_{0,i}^S}$
		$\xleftarrow{\overline{X}, \overline{Y}}$	$\overline{X} = X^s, \overline{Y} = Y^s$
	$K \leftarrow \overline{X}^y$ $\mathsf{AuthG} \leftarrow \mathsf{Hash}_2(A, G, X^\star, \overline{Y}, K)$		
$\xleftarrow{G, \overline{Y}, \mathsf{AuthG}}$			
	$\mathsf{SK} \leftarrow \mathsf{Hash}_1(A, G, X^\star, \overline{Y}, K)$ $\mathsf{accept} \leftarrow \mathsf{true}$		

Fig. 2. Threshold version of the gateway-oriented password-based authenticated key exchange protocol. Since no change is required on the client side with respect to the non-threshold protocol in Figure 1, the client side has been omitted in the diagram.

For a pictorial description of our threshold protocol, please refer to Figure 2.

Security. We now analyze the security of the threshold variant we just presented. To this end, we show a reduction between an adversary $\mathcal{A}$ against the threshold scheme and an attacker $\mathcal{B}$ against the underlying provably-secure GPAKE protocol described in the previous section.

Theorem 7. *If the underlying* GPAKE *protocol is semantically secure and the DDH assumption holds in* $\mathbb{G}$*, then* GTPAKE *is a semantically secure threshold protocol against a static adversary.*

Proof. Our goal is to reduce an adversary $\mathcal{A}$ against the GTPAKE protocol to an attacker $\mathcal{B}$ against the GPAKE protocol. We need to simulate the threshold environment for $\mathcal{A}$ that can corrupt any subset of size at most $k-1$ servers from the environment of the attacker $\mathcal{B}$. We essentially need to simulate the communications among the servers, *i.e.* the decryption parts at the end of the protocol and the proofs of validity.

Let $i_1, \ldots, i_{k-1}$ be the set of corrupted servers. Recall $x_i = F(i) \bmod q$ for all $1 \leq i \leq n$, and $x = F(0) \bmod q$.

To simulate the adversary's view, we simply choose the x_{i_j} belonging to the set of corrupted servers at random from the set $\mathbb{Z}_q$. This allows us to simulate all the messages, proofs of validity and decryption shares coming from the corrupted servers.

Once these values are chosen, the values x_i for the uncorrupted servers are also completely determined modulo q, since we have k points (the $k-1$ points of the corrupted servers and the point 0). The value at point 0 is the decryption value f'^{sx}. However, they cannot be easily computed since $F(0)$ is secret and corresponds to the secret key of the ElGamal scheme. The ElGamal secret key cannot be chosen by the attacker $\mathcal{B}$. Indeed, as all the passwords are encrypted using the ElGamal public key in a public file, $\mathcal{B}$ cannot know it, unless he recovers the passwords and can easily break the semantic security of the GPAKE scheme. However, we can easily compute the decryption parts $g_i^s = f'^{sx_i}$ of the uncorrupted server by using the value f'^{xs}:

$$g_i^s = f'^{sx\lambda_{i,0}^S} \times \prod_{j \in S \setminus \{0\}} g_j^{s\lambda_{i,j}^S}$$

where $S = \{0, i_1, \ldots, i_{k-1}\}$.

For the "proofs of validity", one can invoke the random oracle model for the hash function H to get soundness and perfect zero-knowledge. The soundness is similar to that in Appendix A.

Moreover, the interactive proof system is zero-knowledge against an *honest* verifier since the adversary's view can be simulated without knowing the values x_i. This view includes the values of the random oracle at those points where the adversary has queried the oracle, so the simulator is in complete charge of the random oracle. Whenever, the adversary makes a query to the random oracle, if the oracle has not been previously defined at the given point, the simulator defines it to be a random value, and returns the value to the adversary. When we have to perform a fake proof for $(u_i, \bar{u}_i)$, since the simulator does not know x_i, he chooses at random $c \in \mathbb{Z}_q$ and $z \in \mathbb{Z}_q$ and defines the values of the random oracle at $(p, q, g, \bar{g}, u_i, \bar{u}_i, g^z/u_i^c, \bar{g}^z/\bar{u_i}^c)$ to be c. With all but negligible probability, the simulator has not defined the random oracle at this point before, and so it is free to do so. It is easy to verify that the distribution produced by this simulator is perfect.

Finally, as we need the semantic security of ElGamal encryption scheme, the security is based on the DDH assumption. $\square$

Acknowledgements

The authors would like to thank Raymond Choo for pointing out a flaw in a previous version of our schemes that made them susceptible to attacks by corrupted gateways. Such attacks are not currently taken into account by our security model. The first, third, and fourth authors have been supported in part by France Telecom R&D as part of the contract CIDRE, between France Telecom R&D and Ecole Normale Supérieure. The second author is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Information and Computing Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. This document is report LBNL-58175. isclaimer available at `http://www-library.lbl.gov/disclaimer`.

References

1. Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 65–84, Les Diablerets, Switzerland, January 23–26, 2005. Springer-Verlag, Berlin, Germany.
2. Michel Abdalla and David Pointcheval. Interactive Diffie-Hellman assumptions with applications to password-based authentication. In Andrew Patrick and Moti Yung, editors, *Financial Cryptography 2005*, volume 3570 of *Lecture Notes in Computer Science*, pages 341–356, Roseau, The Commonwealth Of Dominica, February 28 – March 3, 2005. Springer-Verlag, Berlin, Germany.
3. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer-Verlag, Berlin, Germany.
4. Mihir Bellare and Phillip Rogaway. The AuthA protocol for password-based authenticated key exchange. Contributions to IEEE P1363, March 2000.
5. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84, Oakland, CA, May 1992. IEEE Computer Society Press.
6. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In *ACM CCS 03: 10th Conference on Computer and Communications Security*, pages 241–250, Washington D.C., USA, October 27–30, 2003. ACM Press.
7. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. New security results on encrypted key exchange. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *PKC 2004: 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 145–158, Singapore, March 1–4, 2004. Springer-Verlag, Berlin, Germany.
8. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Santa Barbara, CA, USA, August 16–20, 1992. Springer-Verlag, Berlin, Germany.
9. Mario Di Raimondo and Rosario Gennaro. Provably secure threshold password-authenticated key exchange. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 507–523, Warsaw, Poland, May 4–8, 2003. Springer-Verlag, Berlin, Germany.
10. Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security*, 2(3):230–268, August 1999.
11. Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 475–494, Innsbruck, Austria, May 6–10, 2001. Springer-Verlag, Berlin, Germany.
12. Philip D. MacKenzie. The PAK suite: Protocols for password-authenticated key exchange. Contributions to IEEE P1363.2, 2002.
13. Philip D. MacKenzie, Thomas Shrimpton, and Markus Jakobsson. Threshold password-authenticated key exchange. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 385–400, Santa Barbara, CA, USA, August 18–22, 2002. Springer-Verlag, Berlin, Germany.
14. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

A Basic Tools

A.1 Threshold Secret Sharing

Let q be a prime and $1 \le k \le n < q$. Shamir secret sharing over $\mathbb{Z}_q$ is a k-out-of n sharing where any subset of at least k parties can recover the secret, but any subset of strictly less than k parties cannot gain any information about the secret. It is defined as follows: the dealer knows a secret $x \in \mathbb{Z}_q$, chooses $k-1$ random points $f_1, \ldots, f_{k-1} \in \mathbb{Z}_q$, sets $f_0 = x$ and define the polynomial $F(X) = \sum_{j=0}^{k-1} f_j X^j$. For $1 \le i \le n$, let $x_i = F(i) \in \mathbb{Z}_q$ be the i-th share of x. Thanks to the Lagrange equality, we can show that if k shares are revealed, x is completely and can be determined by interpolation. For $S \subset \mathbb{Z}_q$ of cardinality k, any $i \in \mathbb{Z}_q$ and any $j \in S$, there exists an element $\lambda^S_{i,j} = \prod_{j' \in S \setminus \{j\}} (i - j') / \prod_{j' \in S \setminus \{j\}} (j - j') \in \mathbb{Z}_q$ such that

$$F(i) = \sum_{j \in S} \lambda^S_{i,j} F(j) \bmod q$$

A.2 Zero-Knowledge Proof of Equality of Discrete Logarithm.

Let $\mathbb{G}$ be a group of prime order q with generators g and $\bar{g}$. Let $\mathsf{EDLog}_{g,\bar{g}}$ be the language of pairs $(u, \bar{u}) \in \mathbb{G}^2$ such that $\log_g u = \log_{\bar{g}} \bar{u}$. We will use a zero-knowledge proof *of membership* for the language $\mathsf{EDLog}_{g,\bar{g}}$. It is not a proof of knowledge since we only want to prove the correctness of the values computed by the authentication servers. We use the zero-knowledge proof system due to Chaum and Pedersen[8]. Although it happens to also be a proof of knowledge, we will not use this property. We describe here the non-interactive version using the Fiat-Shamir heuristic in the random oracle model.

Let $(u, \bar{u}) \in \mathsf{EDLog}_{g,\bar{g}}$ be given, so there exists $r \in \mathbb{Z}_q$ such that $u = g^r$ and $\bar{u} = \bar{g}^r$.

- The prover chooses $t \in \mathbb{Z}_q$ at random, computes $w = g^t$ and $\bar{w} = \bar{g}^t$. He computes $c = H(p, q, g, \bar{g}, u, \bar{u}, w, \bar{w})$ where H is modeled as a random oracle. Finally, he computes $z = t + rc \bmod q$ and sends to the verifier, (c, z)
- The verifier checks whether the following equality holds

$$c = H(p, q, g, \bar{g}, u, \bar{u}, g^z/u^c, \bar{g}^z/\bar{u}^c)$$

It is well-known that the interactive version of this proof is sound since a cheating prover can be accepted only with probability at most $1/q$. Assume that $(u, \bar{u}) \notin \mathsf{EDLog}_{g,\bar{g}}$, then $u = g^r$ and $\bar{u} = \bar{g}^{\bar{r}}$ where $r \neq \bar{r}$. If a proof is correct, then there exists a unique z such that $w = g^z/u^c$ and $\bar{w} = \bar{g}^z/\bar{u}^c$, then $z - rc = t$ and $z - \bar{r}c = \bar{t}$. Therefore, we get $t - \bar{t} = (\bar{r} - r)c \bmod q$ and since $\bar{r} - r \neq 0 \bmod q$, there is a unique value for $t - \bar{t}$ and so with probability $1/q$, the cheating prover is detected.

Finally, the interactive proof system is zero-knowledge against an *honest* verifier

B Security proofs for GPAKE

B.1 Proof of Theorem 4 (Semantic security)

We are interested in the event S, which occurs if the adversary correctly guesses the bit b involved in the Test-queries. We furthermore consider the gateway (unilateral) authentication: event A is set to true if a client instance accepts, without any gateway partner.

<u>Game $\mathbf{G}_0$:</u> This is the real protocol, in the random-oracle model:

$$\mathbf{Adv}^{\text{ake-ror}}_{\mathsf{GPAKE}}(\mathcal{A}) = 2\Pr[\mathsf{S}_0] - 1 \quad \text{and} \quad \mathsf{Succ}^{\text{G-auth}}_{\mathsf{GPAKE}}(\mathcal{A}) = \Pr[\mathsf{A}_0]. \tag{1}$$

Let us furthermore define the event $\mathsf{S}w/t\mathsf{A} = \mathsf{S} \wedge \neg\mathsf{A}$, which means that the adversary wins the *Real-Or-Random* game without breaking authentication.

<u>Game $\mathbf{G}_1$:</u> In this game, we simulate the hash oracles Hash_1 and Hash_2, as usual by maintaining hash lists $\Lambda_{\mathsf{Hash}_1}$ and $\Lambda_{\mathsf{Hash}'_2}$. In addition to these, we also simulate two private hash functions Hash'_1 and Hash'_2, both mapping $\{0,1\}^\star$ to $\{0,1\}^\ell$, to be used in later games. We also simulate all the instances, as the real players would do, for the Send-queries and for the $\mathsf{Execute}$ and Test-queries.

<u>Game $\mathbf{G}_2$:</u> We cancel games in which some collisions appear on the transcripts $(C, S, X^\star, \overline{Y})$ and on the output of the hash functions Hash_1, Hash_2, and $\mathcal{G}$. Let Ev denote this event. The distance follows easily from the birthday paradox. Note that, in the case of transcripts, at least one element is generated by an honest participant (at least one of them in each of the q_{active} active attacks, and all of them in the q_{exe} passive attacks).

$$\Pr[\mathsf{Ev}_2] \le \frac{q^2_{\text{active}} + q^2_{\mathcal{G}}}{2q} + \frac{q^2_{\text{exe}}}{2q^2} + \frac{q^2_{\text{Hash}_1} + q^2_{\text{Hash}_2}}{2^\ell}. \tag{2}$$

<u>Game $\mathbf{G}_3$:</u> In this game, we show that the success probability of the adversary is negligible in passive attacks via $\mathsf{Execute}$-queries. To do so, we modify the way in which we compute the session key SK and the authenticator AuthG in passive sessions. More precisely, whenever the adversary asks a $\mathsf{Execute}$-query, we compute the session key SK as $\mathsf{Hash}'_1(C\|G\|X^\star\|\overline{Y})$ and the authenticator AuthG as $\mathsf{Hash}'_2(C\|G\|X^\star\|\overline{Y})$, using the private oracles Hash'_1 and Hash'_2. As a result, the values SK and AuthS computed during a passive session become completely independent of the hash functions Hash_1 and Hash_2 and of the Diffie-Hellman key $K_{C/G}$, which are no longer needed in these sessions. Please note that the oracles Hash_1 and Hash_2 are still being used in active sessions.

The games $\mathbf{G}_3$ and $\mathbf{G}_2$ are indistinguishable unless $\mathcal{A}$ queries the hash function Hash_1 or Hash_2 on $(C\|G\|X^\star\|\overline{Y}\|K_{C/G})$, for such a passive transcript: $\mathsf{AskH\text{-}Passive}$. To upper-bound the probability of this event, we consider an auxiliary game $\mathbf{G}_3$', in which the simulation of the players changes — but the distributions remain perfectly identical. Since we do not need to compute $K_{C/G}$ for the simulation of $\mathsf{Execute}$-queries, we can simulate $X^\star$ as $A^{a_1}g^{a_2} \cdot U^{\mathsf{pw}}$, Y as $B^{b_1}g^{b_2}$, and $\overline{Y}$ as Y^s. If event $\mathsf{AskH\text{-}Passive}$ occurs, one can extract $K = \mathrm{CDH}(A^{a_1}g^{a_2}, (B^{b_1}g^{b_2})^s) = \mathrm{CDH}(A^{a_1}g^{a_2}, B^{b_1s}g^{b_2s}) = \mathrm{CDH}(A^{a_1}g^{a_2}, B^{b_1s}) \cdot \mathrm{CDH}(A^{a_1}g^{a_2}, g^{b_2s}) = \mathrm{CDH}(A^{a_1}, B^{b_1s}) \cdot \mathrm{CDH}(g^{a_2}, B^{b_1s}) \cdot \mathrm{CDH}(A^{a_1}, g^{b_2s}) \cdot \mathrm{CDH}(g^{a_2}, g^{b_2s}) = \mathrm{CDH}(A,B)^{a_1b_1s} \cdot B^{a_2b_1s} \cdot A^{a_1b_2s} \cdot g^{a_2b_2s}$ from $\Lambda_{\mathsf{Hash}_1}$ or $\Lambda_{\mathsf{Hash}_2}$:

$$\Pr[\mathsf{AskH\text{-}Passive}_3] \le (q_{\text{Hash}_1} + q_{\text{Hash}_2}) \times \mathbf{Adv}^{\text{cdh}}_{\mathbb{G}}(t + (4q_{\text{exe}} + 4)\tau_e). \tag{3}$$

<u>Game $\mathbf{G}_4$:</u> We now consider passive attacks via Send-queries, in which the adversary simply forwards the messages it receives from the oracle instances. More precisely, we replace Hash_1 by Hash'_1 and Hash_2 by Hash'_2 when computing SK and AuthS whenever the values $(C\|G\|X^\star\|\overline{Y})$ were generated by oracle instances. Note that we can safely do so due to the absence of collisions in the transcript. Like in $\mathbf{G}_3$, the values SK and AuthS computed during such passive sessions become completely independent of the hash functions Hash_1 and Hash_2 and of the Diffie-Hellman key $K_{C/G}$.

As in the previous game, we can unbound the difference in the success probabilities of $\mathcal{A}$ in games $\mathbf{G}_4$ and $\mathbf{G}_3$ by upper-bounding the probability that $\mathcal{A}$ queries the hash function Hash_1 or Hash_2 on $(C\|G\|X^\star\|\overline{Y}\|K_{C/G})$, for such a passive transcript: AskH-Passive-Send. To achieve this goal, we consider an auxiliary game $\mathbf{G}_4$', in which the simulation of the players changes slightly without affecting the view of the adversary. In this simulation, we choose at random one of the $\mathsf{Send}(C,$ "start$''$)-queries being asked to C and we reply with $X^\star = g^{x^\star}$ (hoping that is one of the sessions that the adversary simply forwards the message). On the gateway side, we also change the simulation whenever an instance of the latter receives $X^\star = g^{x^\star}$ as input. In this case, we simply compute the value $\overline{Y}$ as $V^{b_1}g^{b_2}$ and the values SK and AuthG using the private hash functions Hash'_1 and Hash'_2 on input $(C\|G\|X^\star\|\overline{Y})$, respectively. If the event AskH-Passive-Send occurs and our guess for the active session is correct, then we can extract $K = \mathrm{CDH}(g^{x^\star}/U^{\mathsf{pw}}, V^{b_1}g^{b_2}) = (V^{b_1x^\star}g^{b_2x^\star})/(V^{b_1x^\star}\mathrm{CDH}(U,V)^{b_1\mathsf{pw}}$ from $\Lambda_{\mathsf{Hash}_1}$ and $\Lambda_{\mathsf{Hash}_2}$:

$$\Pr[\mathsf{AskH\text{-}Passive\text{-}Send}_4] \le q_{\text{active}} \times (q_{\text{Hash}_1} + q_{\text{Hash}_2}) \times \mathbf{Adv}^{\text{cdh}}_{\mathbb{G}}(t + 2\tau_e). \quad (4)$$

<u>GAME $\mathbf{G}_5$:</u> In this game, we make one of the most significant modifications. We replace the oracles Hash_1 and Hash_2 by the private oracles Hash'_1 and Hash'_2 whenever the input to these queries contains an element that was generated by a client instance but not by a gateway instance. More precisely, if a query $(C\|G\|X^\star\|\overline{Y}\|\mathsf{pw}\|K_{C/G})$ is made to Hash_1 (respectively Hash_2) where only the value $X^\star$ has been simulated (i.e., $\overline{Y}$ was generated by the adversary), then we reply to it using $\mathsf{Hash}'_1(C\|G\|X^\star\|\overline{Y})$ (respectively $\mathsf{Hash}'_2(C\|G\|X^\star\|\overline{Y})$).

Clearly, games $\mathbf{G}_5$ and $\mathbf{G}_4$ are indistinguishable as long as $\mathcal{A}$ does not query the hash function Hash on an input $(C\|G\|X^\star\|\overline{Y}\|K_{C/G})$, where $K_{C/G} = \mathrm{CDH}(X^\star/U^{\mathsf{pw}}, \overline{Y})$, for some execution transcript $(C, G, X^\star, \overline{Y})$. We denote this bad event by AskH-Active*w*C. Thus,

$$|\Pr[\mathsf{A}_5] - \Pr[\mathsf{A}_4]| \le \Pr[\mathsf{AskH\text{-}Active}w\mathsf{C}] \quad |\Pr[\mathsf{S}w/t\mathsf{A}_5] - \Pr[\mathsf{S}w/t\mathsf{A}_4]| \le \Pr[\mathsf{AskH\text{-}Active}w\mathsf{C}] \quad (5)$$

Please note that, at this point, client instances no longer need to know the value of x used to compute $X^\star$ when computing the session key or when verifying an authenticator. Thus, we can simplify the simulation of Send queries of the type $(C^i, \mathbf{start}\ )$ so that the reply $X^\star$ to the latter is simply computed as $g^{x^\star}$ (without using the password).

<u>GAME $\mathbf{G}_6$:</u> In this game, we modify the simulation of the oracles Hash_1 and Hash_2 one last time, in cases where the element $X^\star$ in the query input $(C\|G\|X^\star\|\overline{Y}\|K_{C/G})$ was generated by the adversary, by replacing these oracles with the oracles Hash'_1 and Hash'_2. Since the session key is now being computed with an oracle that is private to simulation, $\Pr[\mathsf{S}w/t\mathsf{A}_6] = \frac{1}{2}$.

Note that, in this game, the exact simulation of the Send oracle is as follows. On a query of type $(C^i, \mathbf{start}\ )$, we reply with $(C, X^\star = g^{x^\star})$ for a random $x^\star \in \mathsf{Z}_p$, if C^i is in the correct state. On a query of type $(G^j, (C, X^\star))$, we reply with $(G, \overline{Y} = g^{\overline{y}}, \mathsf{AuthG})$, where $\mathsf{AuthG} = \mathsf{Hash}'_2(C, G, X^\star, \overline{Y})$ and $\overline{y} \in \mathsf{Z}_p$ is chosen at random, and we set the session key SK_G to $\mathsf{Hash}'_1(C, G, X^\star, \overline{Y})$, if G^j is in the correct state. On a query of type $(C^i, (G, \overline{Y}, \mathsf{AuthG}))$, we first check whether whether $X^\star$ is correct and whether $\mathsf{AuthG} = \mathsf{Hash}'_2(C, G, X^\star, \overline{Y})$, if C^i is in the correct state. If both tests are correct, then we set the session key SK_C to $\mathsf{Hash}'_1(C, G, X^\star,$ $\overline{Y})$. As the following lemma shows, the adversary cannot do much better than simply guessing the password when distinguishing the current experiment from the previous one.

Lemma 8. $|\Pr[\,\textsc{Succ}_0\,] - \Pr[\,\textsc{Succ}_0\,]| \le q_{\text{fake}-\text{G}} \cdot \mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N,q_{\text{Hash}_1}+q_{\text{Hash}_2}}(t, \mathcal{D})$.

Proof. The proof of this lemma is based on a sequence of $q_{\text{fake}-\text{G}} + 1$ hybrid experiments Hyb_j, where j is an index between 0 and $q_{\text{fake}-\text{G}}$. Let i be a counter for number of queries of the form

$(G^k, (C, X^\star))$. That is, we only increment i when the adversary asks a Send query to a gateway instance in which the value $X^\star$ has not been simulated. We define Hyb_j as follows:

- If $i \leq j$, then we processes the current Send query as in Game $\mathbf{G}_6$.
- If $i > j$, then we processes the current Send query as in Game $\mathbf{G}_5$.

It is clear from the above definition that experiments Hyb_0 and $\mathsf{Hyb}_{q_{\text{fake-G}}}$ are equivalent to games $\mathbf{G}_5$ and $\mathbf{G}_6$, respectively. Now let P_j denote the probability of any event Ev in Hyb_j:

$$\Big|\Pr[\mathsf{Ev}_6] - \Pr[\mathsf{Ev}_5]\Big| \leq \sum_{j=1}^{q_{\text{fake-G}}} |P_j - P_{j-1}|. \tag{6}$$

Thus, the lemma will easily follow by showing that the difference in probabilities $|P_j - P_{j-1}|$ between two consecutive hybrid games is at most $\mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N,q_{\text{Hash}_1}+q_{\text{Hash}_2}}(t', \mathcal{D})$.

Let $(G^k, (C, X^\star))$ be the values involved in the crucial j-th Send query to a gateway instance in which the value $X^\star$ has not been simulated. In order to prove the bound for $|P_j - P_{j-1}|$, we first observe that games Hyb_{j-1} and Hyb_j are indistinguishable as long as $\mathcal{A}$ does not query the hash functions Hash_1 or Hash_2 on an input $(C\|G\|X^\star\|\overline{Y}\|K_{C/G})$, where $K_{C/G} = \mathrm{CDH}(X^\star/U^{\mathsf{pw}}, \overline{Y})$ and $X^\star$ and $\overline{Y}$ are the values involved in the crucial j-th Send query. Let$\mathsf{AskH\text{-}Active}w\mathsf{S}$ denote this bad events. Thus, $|P_j - P_{j-1}| \leq \Pr[\mathsf{AskH\text{-}Active}w\mathsf{S}]$.

We now proceed to show that $\Pr[\mathsf{AskH\text{-}Active}w\mathsf{S}] \leq \mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N,q_{\text{Hash}_1}+q_{\text{Hash}_2}}(t, \mathcal{D})$. To do so, consider the following algorithm D_j for the SPCCDH problem in $\mathbb{G}$.

Algorithm D_j. Let $(\mathcal{D})$ be the input given to D_j. D_j starts running $\mathcal{A}$, simulating oracle $\mathcal{G}$ with the help of its input $\mathcal{D}$. Initially, whenever $\mathcal{A}$ asks a Send query, D_j answers to it exactly as one would in game $\mathbf{G}_6$. Note that D_j can easily do so because game $\mathbf{G}_6$ does not require the knowledge of the password to simulate it. D_j will continue to simulate Send queries as in game $\mathbf{G}_6$ until $\mathcal{A}$ asks the j-th $\mathsf{Send}(G^k, (C, X^\star))$-query in which the value $X^\star$ has not been simulated. At this point, D_j outputs $X^\star$ and receives the password Π and a value Y as the input of its second stage. It then sets $\overline{Y} = Y$, computes $K_{C/G}$ and AuthG using the private oracles Hash'_1 and Hash'_2, and returns $(G, \overline{Y} = g^{\overline{y}}, \mathsf{AuthG})$ to $\mathcal{A}$. From all remaining queries, D_j answers to them as as one would in game $\mathbf{G}_5$. Note that it can do so because it knows the value of the password Π. At the end of the simulation, D_j generates its output using the value $K_{C/G}$ of each query to the Hash_1 and Hash_2 oracles such that $X^\star$ and $\overline{Y}$ match the values involved in the crucial j-th Send query.

In order to analyze the success probability of D_j, we first note that the simulation environment provided by D_j to $\mathcal{A}$ is perfect and similar to those of games Hyb_{j-1} and Hyb_j up until the moment in which event $\mathsf{AskH\text{-}Active}w\mathsf{S}$ happens. Moreover, if event $\mathsf{AskH\text{-}Active}w\mathsf{S}$ happens, then the set of queries asked to Hash_1 and Hash_2 contains $\mathrm{CDH}(X^\star/\Pi, \overline{Y})$, the solution to the SPCCDH problem. Thus, $\Pr[\mathsf{AskH\text{-}Active}w\mathsf{S}] \leq \mathbf{Adv}^{\text{spccdh}}_{\mathbb{G},N,q_{\text{Hash}_1}+q_{\text{Hash}_2}}(t, \mathcal{D})$ and the lemma follows. ▌

Game $\mathbf{G}_7$: In this game, we finally evaluate the probability of event $\mathsf{AskH\text{-}Active}w\mathsf{C}$ by considering active attacks against client instances, in which the adversary tries to impersonates the gateway. To do so, we change the simulation of client instances so that the latter rejects all the authenticators sent by the adversary: $\Pr[\mathsf{A}_7] = 0$.

In order to evaluate the distance between the games $\mathbf{G}_7$ and $\mathbf{G}_6$, we consider the probability with which the adversary succeeds in faking the gateway by sending a valid authenticator. To do so, we first notice that since the password is no longer used in the simulation of the oracles, we can postpone choosing its value until the very end of the simulation. Then, since we assume that the size of the dictionary is smaller than 2^ℓ, then for each password pw in the dictionary, there is exactly one valid authenticator (Since collisions on the output of the hash functions have been removed). As a result, the probability that the adversary succeeds in sending a valid authenticator in each session is at most $1/N$. It follows that

$$|\Pr[\mathsf{S}w/t\mathsf{A}_7] - \Pr[\mathsf{S}w/t\mathsf{A}_6]| \leq q_{\text{fake}-\text{C}}/N. \tag{7}$$

Since all sessions in which the adversary is performing an active attack against a client instance results in non acceptance, $\Pr[\mathsf{AskH\text{-}Active}w\mathsf{C}_7] = 0$. Thus, the proof of Theorem 4 follows by combining the results above.

B.2 Proof of Theorem 6 (key privacy)

The proof of key privacy of GPAKE is similar to the one given by Abdalla et al. [1] for their generic 3-party password-based authenticated key exchange protocol. The main difference is that, in our proof, we do not consider Send queries since the server (which is distributed in the real protocol) is assumed to be *honest-but-curious*. As in their case, the key privacy of our protocol follows from the intractability of breaking the Decisional Diffie-Hellman problem in the underlying group $\mathbb{G}$.

Let $\mathcal{A}_{\mathsf{kp}}$ be an adversary against the key privacy of GPAKE with time-complexity at most t, and asking at most q_{exe} queries to its Execute oracle and q_{test} queries to its TestPair oracle. Using $\mathcal{A}_{\mathsf{kp}}$, we can build an adversary $\mathcal{A}_{\mathsf{ddh}}$ for the DDH problem in G as follows.

Let (U, V, W) be the input given to $\mathcal{A}_{\mathsf{ddh}}$. $\mathcal{A}_{\mathsf{ddh}}$ first chooses the passwords for all users in the system according to the distribution of Dict. It also chooses a bit b at random that is used to answer queries to the TestPair oracle. It then starts running $\mathcal{A}_{\mathsf{kp}}$ giving all the password of all users to it. $\mathcal{A}_{\mathsf{ddh}}$ will use the classical random self-reducibility of the Diffie-Hellman problem to introduce its input triple in the answers to Execute and TestPair.

To simulate the $\mathsf{Execute}(C^i, G^j)$ oracle, $\mathcal{A}_{\mathsf{ddh}}$ first chooses random values a_0, a_1, b_0, b_1, and s in $\mathbb{Z}q$. Then, it computes $X = U^{a_0} g^{a_1}$ and $X^\star = X \times \mathcal{G}(\mathsf{pw})$, by using the previously-chosen password and simulating the random oracle $\mathcal{G}$ in the usual way, and the values $Y = V^{b_0} g^{b_1}$, $\overline{X} = X^s$, and $\overline{Y} = Y^s$. Next, $\mathcal{A}_{\mathsf{ddh}}$ sets the Diffie-Hellman key K to $W^{a_1 b_1 s} \cdot U^{a_1 b_2 s} \cdot V^{a_2 b_1 s} \cdot g^{a_2 b_2 s}$ and computes $\mathsf{AuthG} = \mathsf{Hash}_2(C \| G \| X^\star \| \overline{Y} \| K)$ and the session key $\mathsf{SK} = \mathsf{Hash}_1(C \| G \| X^\star \| \overline{Y} \| K)$. Finally, $\mathcal{A}_{\mathsf{ddh}}$ gives $(X^\star, Y, \overline{Y}, \overline{X}, \mathsf{AuthG})$ to $\mathcal{A}_{\mathsf{kp}}$.

To simulate the $\mathsf{TestPair}(C^i, G^j)$, $\mathcal{A}_{\mathsf{ddh}}$ first checks whether this same query has been asked before and gives the same response if that is the case. Otherwise, $\mathcal{A}_{\mathsf{ddh}}$ checks whether C^i and G^j are indeed partners, and then gives to $\mathcal{A}_{\mathsf{kp}}$ either the session key SK if $b = 0$ or a random value in $\mathbb{G}$ if $b = 1$.

To analyze the success probability of $\mathcal{A}_{\mathsf{ddh}}$, first consider the case in which the triple (U, V, W) is a true Diffie-Hellman triple. Then, in this case, one can see that simulation of the $\mathcal{A}_{\mathsf{kp}}$ oracles is perfect. Hence, the probability that $\mathcal{A}_{\mathsf{ddh}}$ outputs 1 is exactly $\frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\text{ake}-\text{kp}}_{\mathsf{GPAKE},\mathsf{Dict}}(\mathcal{A}_{\mathsf{kp}})$. On the other hand, when (U, V, W) is a random triple, the keys used to answer TestPair queries are all random and independent as a result of the random self-reducibility property of the Diffie-Hellman problem. Hence, no information on b is leaked through TestPair queries and the probability that $\mathcal{A}_{\mathsf{ddh}}$ outputs 1 is exactly $\frac{1}{2}$ in this case. The proof of Theorem 6 follows from the fact that $\mathcal{A}_{\mathsf{ddh}}$ has time-complexity at most $t + 8(q_{\text{exe}})\tau_e$, due to the additional time for the computations of the random self-reducibility.

C Proof of Lemma 2

In this section, we show that the Set Password-based Computational Diffie-Hellman SPCCDH problem is equivalent to the (basic) computational Diffie-Hellman problem CDH: For proving this relation, one simply applies the splitting lemma [14]:

Lemma 9 (Splitting Lemma). *Let* $S \subset A \times B$ *such that* $\Pr[(a,b) \in S] \geq \alpha$*. For any* $\beta < \alpha$*, define*

$$T = \left\{ (a,b) \in A \times B \;\middle|\; \Pr_{b' \in B}[(a,b') \in S] \geq \alpha - \beta \right\}$$

Then *(i)* $\Pr[T] \geq \beta$ *(ii)* $\forall (a,b) \in T, \Pr_{b' \in B}[(a,b') \in S] \geq \alpha - \beta$*.*

Let $\mathcal{A}$ be an adversary against the SPCCDH problem, with success probability $\alpha = 1/N + \varepsilon$. Then, we can use the splitting lemma, with $\beta = \varepsilon/2$, on

$$A = \{(\omega, X, \mathcal{D})\} \text{ and } B = \{1, \ldots, N\} \approx \mathcal{D}.$$

Our adversary $\mathcal{B}$ receives as input a random CDH instance (U, X). It chooses a random tape ω for $\mathcal{A}$, as well as N random distinct exponents $u_i \in \mathsf{Z}_p$. It defines $U_i = U^{u_i}$, which specifies the dictionary $\mathcal{D}$: with probability greater than $\varepsilon/2$, the success probability is greater than $1/N + \varepsilon/2$, over the probability space $B = \{1, \ldots, N\}$. It is thus a multiple of $1/N$, not smaller than $1/N + \nu$, where ν is the maximum in $\{1/N, \varepsilon/2\}$. One first simply runs $\mathcal{A}$ with a random k, and with probability greater than $1/N + \nu$, one gets a first set $\mathcal{S}_1$ with $K = \mathrm{CDH}(X/U_k, Y) = \mathrm{CDH}(X/U^{u_k}, Y)$. One runs $\mathcal{A}$ again, with another random $k' \neq k$, and with probability greater than ν, one gets a second set $\mathcal{S}_2$ with $K' = \mathrm{CDH}(X/U_{k'}, Y) = \mathrm{CDH}(X/U^{u_{k'}}, Y)$. Then, $\mathrm{CDH}(X, U) = (K/K')^{1/(u_k - u_{k'})}$. By choosing two elements at random in $\mathcal{S}_1$ and $\mathcal{S}_2$, one gets $\mathrm{CDH}(X, U)$ with probability $1/s^2$.