Anonymous and Transparent Gateway-based Password-Authenticated Key Exchange

Michel Abdalla, Malika Izabachène, David Pointcheval

An extended abstract of this paper appeared in
Proceedings of the 7th International Conference on Cryptology and Network Security (CANS '08)
December 2–4, 2008, Hong-Kong, China – M. Franklin, L. Hui and D. Wong Eds. Springer-Verlag, LNCS 5339, pages 133–148.

Anonymous and Transparent Gateway-based Password-Authenticated Key Exchange

Michel Abdalla, Malika Izabachène, and David Pointcheval

Ecole Normale Supérieure, CNRS, INRIA, France

Abstract. In Asiacrypt 2005, Abdalla *et al.* put forward the notion of gateway-based password-authenticated key exchange (GPAKE) protocol, which allows clients and gateways to establish a common session key with the help of an authentication server. In addition to the semantic security of the session key, their solution also provided additional security properties such as password protection with respect to malicious gateways and key privacy with respect to curious authentication servers. In this paper, we further pursue this line of research and present a new and stronger security model for GPAKE schemes, combining all above-mentioned security properties. In addition to allowing a security proof for all these security properties, the new security model has also other advantages over the previous one such as taking into account user corruptions. After describing the new security model, we then present a new variant of the GPAKE scheme of Abdalla *et al.* with similar efficiency. Like the original scheme, the new scheme is also *transparent* in that it does not differ significantly from a classical 2-PAKE scheme from the point of view of a client. Finally, we also show how to add client anonymity with respect to the server to the basic GPAKE scheme by using private information retrieval protocols.

1 Introduction

1.1 Motivation

To address practical scenarios in which the service provider is actually composed of two distinct entities, one being the direct interlocutor of the client and the other being a back-end server capable of checking the identity of the client, Abdalla *et al.* [1] put forward the notion of gateway-based authenticated key exchange. A gateway-based authenticated key exchange [1] is a three-party protocol, which provides a client C and a gateway G with a common session key with the help of an authentication server S, which authorizes or not the access for the client. Among the various means of authentication that can be considered, the most interesting one from a practical point of view is the password-based setting in which a simple human-memorizable secret, called a password, is used for authentication.

Due to the low entropy of passwords, gateway-based password-authenticated key exchange protocols may be subject to exhaustive search attacks, also known as *dictionary attacks* [5, 7, 10, 18, 21], in which the adversary tries to break the security of the scheme by trying all possible values for the password. In these attacks, which can be either online or off-line depending on whether the attacker needs to interact with the system in order to test whether a password guess is correct, the success probability of an attacker can be significantly high. Since online dictionary attacks cannot be avoided, the main goal of gateway-based authenticated key exchange protocols based on passwords is to show that off-line dictionary attacks are not possible. As in other password-based protocols, this is done by showing that, after k active attempts, the success probability of an adversary of impersonating one of parties or distinguishing a session key from a random key is at most $O(k/N)$, where N is the size of the dictionary and the hidden constant is preferably 1, and not much more than that as could be the case with partition attacks [9].

1.2 Related Work

Unfortunately, security against off-line dictionary attacks may not be sufficient since some of the parties involved in the protocol may be malicious. In particular, it may be possible for a malicious gateway to gain information about the values of user passwords and for the malicious authentication servers to learn the value of the session keys. To overcome this problem, Abdalla,

Chevassut, Fouque, and Pointcheval [1] proposed a new setting for gateway-based systems, termed gateway-based password-authenticated key exchange (GPAKE), which protects both the session keys and the passwords. In their work, in addition to the usual notion of *semantic security* outlined above, two other security goals were considered: key privacy with respect to servers and password protection with respect to gateways. The former says that the session key should remain indistinguishable from random, even with respect to a honest-but-curious server that knows the passwords of all the users. The latter states that the gateway should not learn any information about the passwords of clients, from the authentication server.

1.3 Contributions

In this paper, we further investigate the line of research initiated by Abdalla *et al.* [1] on GPAKE schemes. In a first step, we provide in Section 2.2 a new and stronger security model which captures all of the above security notions in a single security game. The new model has several advantages over the previous one. First, the new model does not require a separate proof for each security property. Second, the new model also allows for corruptions of participants, thus dealing with the issue of forward secrecy. Third, in relation to the key privacy with respect to the server, the new security model also extends the class of sessions for which the session keys are private to the adversary, in particular allowing some sessions to remain fresh even after the corruption of a player.

After describing the new security model, we then present in Section 3 a new scheme based on the GPAKE scheme of Abdalla *et al.* [1]. Like the original scheme, our new GPAKE scheme does not require too much additional computational load for the client when compared to a classical 2-PAKE scheme. This is the so-called *transparency* property: from the client point of view, the scheme should not differ significantly from a classical 2-PAKE scheme between 2 parties. Though the security guarantees provided by the new scheme are stronger, the latter is only slightly less efficient than the original scheme proposed in [1]. The complexity assumptions used in the proof of security of the new scheme are also similar to those used by the original scheme in [1]. In both cases, the proof of security is in the random oracle model.

An additional feature considered in [1] was password privacy with respect to servers, in which users' passwords are kept secret even from servers. To achieve this goal, the responsibility for authenticating users was distributed among several servers using standard techniques in threshold-based cryptography. Such a technique can be applied to our new protocol too. In Section 4, we address a different and perhaps more crucial privacy concern: the possibility that servers can log connections and profile users. We would thus like to preserve the *anonymity* of the clients and ensure that the connections are unlinkable, in the same vein as [11]. Towards this goal, we show in Section 4 how to add client anonymity to the scheme in Section 3 by using Private Information Retrieval protocols (PIR) [15, 16]. The use of PIR is not new, but it shows a new feature of our GPAKE protocol: it can be efficiently interfaced with a PIR. Furthermore, this new feature can be used in conjunction to the previous password privacy with respect to the servers.

Finally, it is important to note that, since we designed our protocol with client anonymity in mind, it is not possible for a server to distinguish an honest authentication request by a client from an online impersonation attack by a malicious gateway. That is, the server cannot detect *online dictionary attacks*. As a result, clients' passwords should be renewed more often in the present scheme than in the case of standard 2-PAKE schemes. In cases where this is not possible, one should use instead a protocol without client anonymity and in which the server can detect online dictionary attacks.

2 Security Model

2.1 Notation

In the three-party-protocol, each participant will be denoted as $U \in \mathcal{U}$, which can be either a client, a gateway or the (authentication) server. In the password-based scenario, each client C holds a password $pw_C \in \mathcal{D}$, of small entropy and the server manages a database $\mathtt{DB}$, with all the clients' passwords. Here, we suppose that there is only one (authentication) server.

We denote by $\mathcal{C}$ and $\mathcal{G}$ the sets of the clients and the gateways respectively and by S the server. We thus have $\mathcal{U} = \mathcal{C} \cup \mathcal{G} \cup \{S\}$. Since any party can be involved concurrently in several executions, several *instances* of a party can be activated: we denote by $\mathcal{I}$ the set of instances and by U_i^s the s-th instance of a participant U_i. When there is no ambiguity, we omit to precise the instance of the participant, for the sake of clarity in the writing. When C_i^s and G_j^t are engaged in a conversation to compute a common session key, in case of success, we denote the session key $sk_{i,j}^{s,t} = sk_{C_i^s} = sk_{G_j^t}$, and we say that C_i^s and G_j^t are *partners* in this session (see Definition 2).

Definition 1 (Gateway-based Authenticated Key-Exchange).
A gateway-based key-exchange protocol is specified by four polynomial-time algorithms $P = (\mathtt{LL}, \mathtt{CLIENT}, \mathtt{GATEWAY}, \mathtt{SERVER})$:

- $\mathtt{LL}$ specifies the initial distribution of the long-lived keys. It takes as input a security parameter k;
- $\mathtt{CLIENT}$ specifies how a client C behaves; It takes as input an instance C^s, a state s_C, the identity of the sender, and a message m. It outputs the message the client C should send as answer;
- $\mathtt{GATEWAY}$ specifies how a gateway G behaves; It takes as input an instance G^s, a state s_G, the identity of the sender, and a message m. It outputs the message the gateway G should send as answer;
- $\mathtt{SERVER}$ specifies how S behaves; It takes as input an instance S^s, a state s_S, the identity of the sender, and a message m. It outputs the message the server S should send as answer.

In the definition above, the specific format of the input message m will depend on the specific protocol being analyzed. $\mathtt{LL}(k)$ is a probabilistic polynomial-time algorithm, which returns the long-lived keys for all the participants. In our particular case of password-based authentication for the client, and symmetric private and authenticated channels between the gateways and the server, they consist in:

- a password pw for each client (given to the server too);
- a symmetric key $\mathsf{SecureChan}$–S–G for each gateway-server pair to provide a secure channel (with privacy, integrity and origin of the messages).

2.2 Security Model

In this section, we describe how an adversary will be allowed to interfere in the protocol. Our model *extends* the previous work on three-party and gateway-based key-exchange protocols [1,2] by considering together the notions of:

- key privacy (semantic security of the session key) with respect to the server, and even under corrupted players (which includes forward-secrecy);
- client's password protection with respect to the gateway;
- client privacy (anonymity) with respect to the server.

The former extends the classical notion, and namely it includes the server in the adversary list. This also justifies the quite new latter notion of anonymity with respect to the server, that we do not trust, but just for authenticating a valid client, and authorizing the connection. As in [2], we do not trust the gateway, and thus the password must not be known to it.

We would like to emphasize that the stronger notion of semantic security that we study is of independent interest. It indeed extends usual security models for 2-party password-based key exchange protocols [5, 10]: we consider a broader notion of freshness, which is crucial to define which sessions have to be protected or not. Usually, as soon as a user is corrupted, all the new sessions are not fresh. However, if the real users play the protocol, the session key may still be secure in practice, we will introduce that in our security model (this is definitely stronger than what can be achieved in the universal composability framework [12, 13] as we see later).

Oracle Queries. As in [2], we adopt the Real-or-Random (RoR) security model, which means that the adversary interacts with the protocol via oracle queries (in any order it wants — concurrent executions—, and should not be able to distinguish truly random keys from the real keys. As it has been proved in [2], this model is stronger than the Find-then-Guess (FtG) security model, where the adversary only has to distinguish a random key from a real one, but in one session only. The oracle queries are the following ones:

- $\mathsf{Execute}(C_i^r, G_j^s, S^t)$: This query models passive attacks in which the adversary asks for an honest execution of the entire protocol. The adversary gets the whole transcript resulting from the communication between C_i^r, G_j^s and S^t.
- $\mathsf{Send}(U_i^r, U; m)$: This query models an active attack, where the adversary chooses the message m it sends to the instance U_i^r, where $U_i \in \mathcal{C} \cup \mathcal{G} \cup \{S\}$ in the name of U. The adversary gets back the message U_i should produce upon receiving such a message m. If the message is invalid or if U_i^r is not in a waiting state, the query is ignored.
  Because of the private and authenticated channels between the gateways and the server, if the gateway G is not corrupted (the flag $\mathsf{corrupt}_G$ is false), the recipient does not accept a message that has not really been sent by the legitimate sender.
- $\mathsf{Reveal}(U_j^s)$: if the session key for instance U_j^s has ever been defined, then the answer is the actual session key $sk_{U_j^s}$, otherwise, the answer is $\perp$.

Intuitively, the Execute-query models passive observations of transcripts and Send-queries model active attacks against some honest players. They can be combined in an attack game. Then, sessions obtained via Execute-queries are called "passive sessions", whereas sessions obtained via Send-queries are called "active sessions". Of course, Execute-queries can be simulated by a sequence of Send-queries, but such a sequence of Send-queries is counted as an "active session". In the password-based setting, our goal is to prove that the on-line dictionary attack is the best one: passive sessions do not leak any information about the password, an active session can help the adversary to test only one password.

The Reveal-query models a misuse of the established session key, and thus the leakage of information: session keys should not reveal any information about the password.

Semantic Security. One goal of our protocol is to ensure that the session keys established between C and G remain completely unknown to S and to any other party: *session key privacy*, also known as *semantic security*. Indeed, we want that the adversary could not get any relevant information on the fresh session keys (not obviously revealed, see the freshness notion below) from the above queries. To model the capability of the adversary to guess some information on the session key, we define an additional Test-query. for a random bit b as follows:

- $\mathsf{Test}(U_j^s)$: If no session key is defined for instance U_j^s or if instance U_j^s is not fresh (see notion of freshness below), then this query is answered by $\perp$. If this query has already been asked, then it outputs the same answer. Otherwise if $b = 1$, it outputs the real session key (from

$\mathsf{Reveal}(U_j^s)$), and if $b = 0$, it outputs a random one of same size. Finally, the flag $\mathsf{test}_{U_j^s}$ becomes true.

Note that the answer of the oracle is independent of the number of queries the adversary asks. Indeed, the oracle always answers the same way: all the session keys obtained via Test-queries are either all real or all random, according to the bit b. Furthermore, as soon as the flag $\mathsf{test}_{U_j^s}$ is set to true, no $\mathsf{Reveal}(U_j^s)$ can be asked.

Corruption. We say that a participant U is corrupted and we set $\mathsf{corrupt}_U$ to true if one of the $\mathsf{Corrupt}$-queries has been asked (it is initially set to false). Note that we consider weak corruptions only, where the long term secrets are revealed, but not the internal states.

- $\mathsf{Corrupt}(C_i)$: This query models corruption of the client in which the adversary learns the password of client C_i. We then set $\mathsf{corrupt}_{C_i} \leftarrow \mathsf{true}$.
- $\mathsf{Corrupt}(G_j)$: This query models corruption of a gateway in which the adversary gets access (read/write) to the secure channel between the gateway G_j and the server. We then set $\mathsf{corrupt}_{G_j} \leftarrow \mathsf{true}$.
- $\mathsf{Corrupt}(S)$: This query models corruption of the server S in which the adversary learns all the passwords stored into the server, i.e, pw_C for all the clients C, and gets access to the secure channels between the server and all the gateways (since we assumed a symmetric protection tool). We then set $\mathsf{corrupt}_C \leftarrow \mathsf{true}$ and $\mathsf{corrupt}_G \leftarrow \mathsf{true}$ for all the clients C and all the gateways G.

Partnering. We use the notion of partnering based on session identifiers (sid), as defined in [5]. In particular, the value of sid is taken to be the partial transcript of the communication between the client and the gateway before the session key has been accepted.

Definition 2. Two instances C_i^s and G_j^t are said to be partners if and only if the following four conditions hold:

1. Both C_i^s and G_j^t have accepted;
2. Both C_i^s and G_j^t share the same sid;
3. The partner for C_i^s is G_j^t and vice-versa;
4. No instance other than C_i^s accepts with partner G_j^t and vice-versa.

Freshness. The goal of the adversary in the Real-or-Random game is to guess the bit b used during Test-queries. However, this is clear that in some cases, the adversary trivially knows the actual session key. And, then, the Test-query answer would immediately lead to the bit b value, whereas the adversary did not really break the semantic security. For example, if the adversary asks a Reveal-query and a Test-query to the same session.

We thus need to restrict the use of the Test-queries: they must be asked to *fresh* sessions/instances only.

Informally, we say that an instance is not fresh, and fresh_{U^s} is thus set to false, when the adversary could trivially distinguish a random session key from a real one, or when no key exists yet:

- if U has not accepted. Note that it will make no sense to consider this case since no session key has been defined: an instance that has not accepted is not fresh. Hence the initialization of the fresh-tags to false.
- if U has been asked a Reveal-query, any Test-query to U or its partner clearly reveal b. Such a Reveal-query thus flips the freshness status of the instance and its partner to false.

These two restrictions are classical, and in order to capture the forward-secrecy, a corruption that happens after the end of the protocol does not affect the freshness status. We consider this stronger forward-secrecy notion in this paper, contrarily to [1].

Moreover, in previous freshness definitions, as soon as a party was corrupted, the sessions initiated afterward were automatically not fresh (this is also the case in the UC security model, since a corrupted player is under the control of the adversary, and cannot play on its own). Here are some reasons for considering some sessions initiated after a corruption as not fresh:

- if C is corrupted, the adversary could trivially break the semantic security by playing the role of the gateway and the server, against the client: he chooses all gateway and server randomness to compute the common session key;
- if C is corrupted, then the adversary knows the client's password then he could play the role of the client, against the gateway;
- if G is corrupted, the adversary can play the role of the gateway, with both the server and the client.

However, even if everybody is corrupted but the adversary is passive during this session (an Execute-query or simple forwarding via Send-queries), there is no reason to mark this session as unfresh. In practice, even if the long term key of a player has been stolen, the latter can be sure for some session that it is talking with an honest player: it is useful to know that in such a case, the session key will be secret, hence the freshness of such a session.

Definition 3. When a player U accepts (either C_i^s or G_j^t), if both this party and its partner are not corrupted; or, if all the messages received by this party were generated by an honest user (only oracle-generated-messages —see below— or a passive session —through an Execute-query); then $\mathsf{fresh}_U \leftarrow \mathsf{true}$, else $\mathsf{fresh}_U \leftarrow \mathsf{false}$. Later, a Reveal-query can flip the status from true to false, as explained earlier.

Oracle-Generated Messages. When an instance U^s is asked a Send-query for a message m, so that m has been answered by a Send-query, then we say that this is an oracle-generated-message.

We write $\mathsf{OG}(U^s)$ when U^s receives an oracle-generated-message. More generally, we write $\mathsf{OG}(U^s, n)$, when the n-th flow (in the protocol) received by U^s, is an oracle-generated-message.

Definition 4. We say that m is an oracle-generated-message if there exists an instance U_j^s, for $s \in \mathcal{I}$ and a participant $U_i \in \mathcal{U}$ such that $m = \mathsf{Send}(U_j^s, U_i; m')$ for some message m'.

Due to the fact that we assume a secure channel, and thus authenticated, between the gateways and the server, all non-oracle-generated messages between the server and a non-corrupted gateway will be rejected with overwhelming probability, if we choose an appropriate symmetric encryption mechanism. Indeed, if the gateway is not corrupted no message can be correctly encrypted to and from it, when communicating with the server.

Secure Protocol. We consider an adversary $\mathcal{A}$, against a protocol P, which has access to Execute, Send, Reveal-queries, as defined above, as well as to Test-queries to *fresh instances* only. Let Succ be the event in which the adversary guesses correctly the bit b determining the behavior of the Test-query (whether it outputs the real session key or a random one). We define the AKE-advantage of the adversary $\mathcal{A}$ in breaking the semantic security of the protocol P by : $\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) = 2 \cdot \Pr[\mathsf{Succ}] - 1$.

Definition 5. We say that a password-authenticated key-exchange protocol P is secure if for every polynomial time adversary $\mathcal{A}$ that interacts actively with at most q instances,

$$\mathsf{Adv}_P^{\mathsf{ake}}(\mathcal{A}) < \frac{c \cdot q}{N} + \mathsf{negl}(),$$

where N is the size of the dictionary where the passwords are uniformly drawn, and c is a small constant (ideally 1, but a larger constant may appear because of the proof details).

Note that the security notions of semantic security, key privacy with respect to the server, forward-secrecy, and password protection, as defined by [1] are implied by the above security definition. For the three first notions, this is clear since they are modeled by the Test-query. About the password-protection, if a corrupted gateway could learn some information about the password, then this information could be used by the adversary to perform a faster on-line dictionary attack.

3 Our new GPAKE Protocol

3.1 Description of our Scheme

In this section, we describe our new GPAKE protocol, which is a slight variant of [1]. Let $(\mathbb{G}, g, q)$ be the description of a cyclic group $\mathbb{G}$ of order q generated by g. Let ℓ be the security parameter. We need several hash functions $\mathcal{G}$, $\mathcal{H}_1$, and $\mathcal{H}_2$:

$$\mathcal{G} : \mathcal{U}^2 \times \mathcal{D} \mapsto \mathbb{G}, \quad \mathcal{H}_1 : \mathcal{U}^2 \times \mathbb{G}^3 \mapsto \{0,1\}^\ell, \quad \mathcal{H}_2 : \mathcal{U}^2 \times \mathbb{G}^3 \mapsto \{0,1\}^\ell.$$

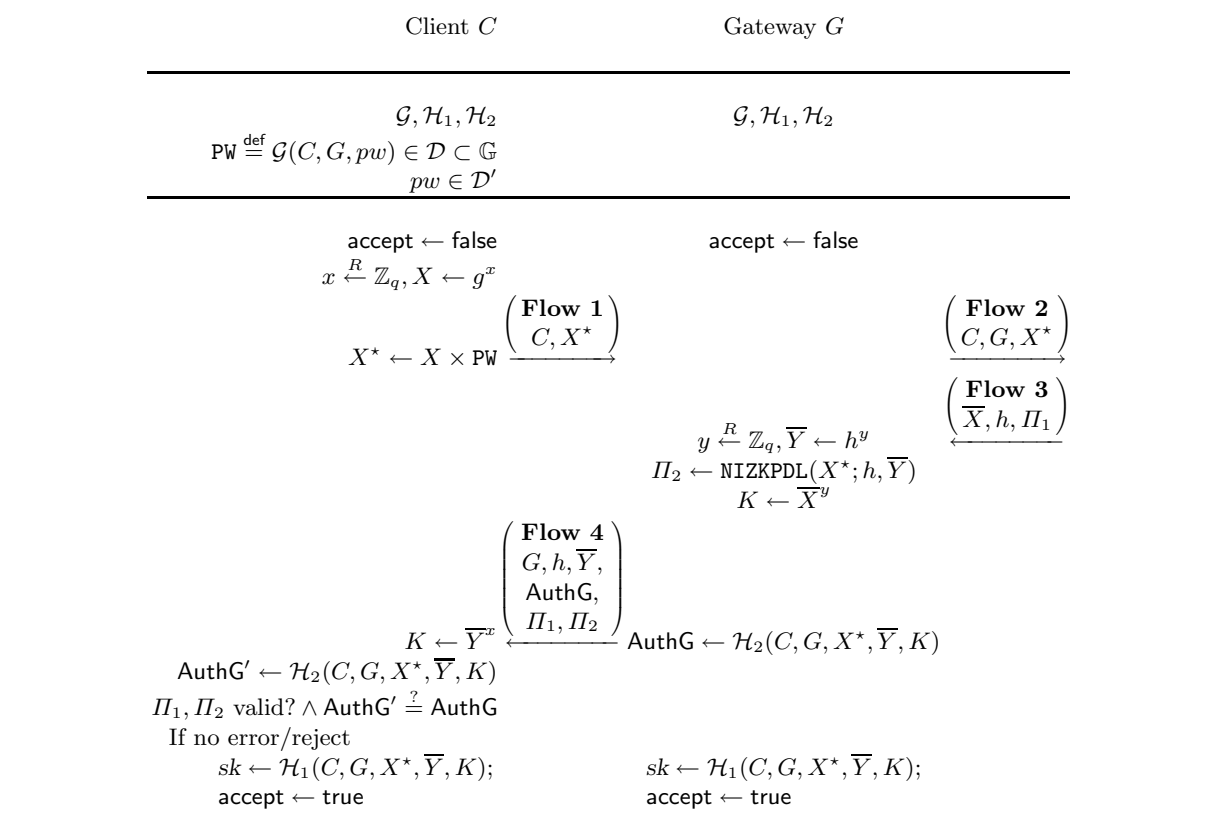
They will be modeled by random oracles in the security analysis. From each password $pw \in \mathcal{D}'$, we define $\mathtt{PW} \stackrel{\mathsf{def}}{=} \mathcal{G}(C, G, pw) \in \mathcal{D} \subset \mathbb{G}$, assumed to be the authentication means between the client C and the gateway G, with which the client wants to establish a secure channel (*transparency* means that the client does not need to know whether the gateway can compute everything by itself or needs some help from outside). The client knows pw, and the gateway will ask some help from the server who knows the PW's, since the storage limitation is not as strong for him. Such a random generation of the actual common secret PW (hence the use of a random oracle $\mathcal{G}$) is crucial for the PCDDH-assumption below. The new protocol still consists of four messages exchange between the client, the gateway and the server as in [1]. To achieve the stronger notion of freshness, which may include certain sessions with corrupted players, we need two additional zero-knowledge proofs of knowledge of the discrete logarithms of h and $\overline{Y}$ with respect to bases g and h (see Section 3.2). The complete description can be found in Figure 1, where NIZKPDL signatures of knowledge are described in the next section. Another difference with respect to the protocol of [1] is that, upon the reception of $X^\star$ and the client identity string C, the gateway just forwards these two elements to the server. Then, the gateway chooses the exponent y and computes $\overline{Y}$ and K at the same time. This allows us to introduce anonymity as we see later in Section 4.

3.2 Zero-Knowledge Proof of Knowledge of Discrete Logarithm

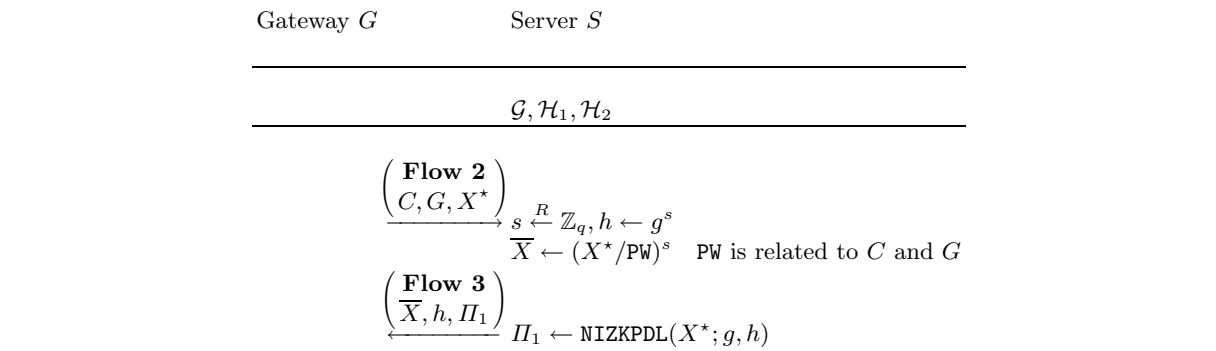
In our protocol, we need non-interactive zero-knowledge proofs of knowledge of discrete logarithms. This will actually be a signature of knowledge: we denote by $\mathtt{NIZKPDL}(m; g, h)$ the signature of knowledge of the discrete logarithm of h in basis g on the message m. This is the Schnorr's signature [24, 25], proved secure in the random-oracle model [22, 23]. Granted the forking-lemma, when such a valid proof is generated by the adversary, extraction is possible, operating one rewind.

More precisely, let us describe the zero-knowledge proofs Π_1 and Π_2 used in our scheme, on the message $X^\star$. Let $\mathbb{G}$ be a cyclic group of order q generated by g. We use a non-interactive version of the Schnorr's proof system presented in [24, 25] to which we apply the Fiat-Shamir transformation [17] in the random-oracle model [6, 22, 23]:

- Public data: $h = g^s$, description of $(\mathbb{G}, g, q)$
- Witness: the exponent s
- Proof of knowledge of the witness: the prover chooses a random exponent $\alpha \in \mathbb{Z}_q$ and sets $u = g^\alpha$. He computes $c = H(X^\star, g, h, u)$ and $v = \alpha - cs \mod q$. He then sends the proof (c, v) to the verifier.



The communication channel between the client and the gateway is not authenticated nor private



The communication channel between the gateway and the server is secure (privacy, integrity and origin of the messages)

Fig. 1. Our new GPAKE

– Verification of the proof: $c \stackrel{?}{=} H(X^\star, g, h, g^v h^c)$.

The forking lemma [22, 23] shows that, in an execution in which a valid proof is generated, in polynomial time, it is possible to extract the witness, from one of the rewinds with a different random choice for the random oracle, but with any non-negligible (as small as required) probability.

Note however that during a rewind we have to make sure that no other extraction is needed, which would imply another rewind, and so. An exponential complexity could happen if several extractions are needed. But in our analysis, only one extraction is needed.

3.3 Computational Assumptions

For the security of our scheme, we need the following computational assumptions, which are either quite classical, or at least already used in the past [4]. The last one could be shown to hold in the generic-group model [26].

Computational-Diffie-Hellman Assumption. Let $(\mathbb{G}, g, q)$ be a represented group. The CDH-assumption states that given two elements $A = g^a$ and $B = g^b$, where a and b are drawn at random from $\mathbb{Z}_q$, it is hard to compute $C = A^b = B^a = g^{ab}$. More precisely, given the experiment,

$$\begin{array}{l}\mathbf{Exp}^{\mathsf{cdh}}\ (\mathcal{A})\\ \quad a, b \stackrel{R}{\leftarrow} \mathbb{Z}_q, A \leftarrow g^a, B \leftarrow g^b, C \leftarrow g^{ab}\\ \quad C' \leftarrow \mathcal{A}(A, B)\\ \quad \text{if } C = C', \text{output } 1, \text{else output } 0\end{array}$$

the advantage of $\mathcal{A}$ in breaking the CDH-problem, defined by $\mathsf{Adv}^{\mathsf{cdh}}\ (\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{cdh}}\ (\mathcal{A}) = 1]$, in reasonable time, is negligible.

Decisional-Diffie-Hellman Assumption. The DDH-assumption states that the two distributions $(g, A = g^a, B = g^b, C = g^{ab})$ and $(g, A = g^a, B = g^b, C' = g^c)$, where a, b, c are drawn at random from $\mathbb{Z}_q$ are computationally indistinguishable. As previously done, we can define an experiment:

$$\begin{array}{l}\mathbf{Exp}_d^{\mathsf{ddh}}\ (\mathcal{A})\\ \quad a, b, c \stackrel{R}{\leftarrow} \mathbb{Z}_q, A \leftarrow g^a, B \leftarrow g^b\\ \quad \text{if } d = 0, \text{set } C \leftarrow g^c, \text{else set } C \leftarrow g^{ab}\\ \text{output } d' \leftarrow \mathcal{A}(A, B, C)\end{array}$$

The advantage of $\mathcal{A}$ in deciding the DDH-problem in reasonable time is negligible, where:

$$\mathsf{Adv}^{\mathsf{ddh}}\ (\mathcal{A}) = \left|\Pr[\mathbf{Exp}_1^{\mathsf{ddh}}\ (\mathcal{A}) = 1] - \Pr[\mathbf{Exp}_0^{\mathsf{ddh}}\ (\mathcal{A}) = 1]\right| = \mathsf{negl}().$$

Password-Based Chosen-Basis Diffie-Hellman Assumption. The following problem is a variant from [4]. Let $(\mathbb{G}, g, q)$ be a represented group. The following experiment, $\mathbf{Exp}_b^{\mathsf{pcddh}}\ (\mathcal{A}, \mathcal{D})$ defines how we simulate the interaction between the Password-based Chosen-basis Decisional Diffie-Hellman challenger, or PCDDH-challenger, and the adversary $\mathcal{A}$, where $\mathcal{D}$ is a dictionary of N random and independent elements in $\mathbb{G}$. In a `find`-stage, the adversary chooses a basis X. It then receives back $\mathtt{PW}$, X' and Y, defined as follows:

$$\mathtt{PW} \stackrel{R}{\leftarrow} \mathcal{D}, \quad s_0, s_1 \stackrel{R}{\leftarrow} \mathbb{Z}_q, \quad X' \leftarrow (X/\mathtt{PW})^{s_b}, \quad Y \leftarrow g^{s_0}.$$

Then, the adversary has to guess the bit b:

$$\begin{array}{l}\mathbf{Exp}_b^{\mathsf{pcddh}}\ (\mathcal{A}, \mathcal{D})\\ \quad (X, s) \leftarrow \mathcal{A}(\mathtt{find}, \mathcal{D})\\ \quad \mathtt{PW} \stackrel{R}{\leftarrow} \mathcal{D}, s_0, s_1 \stackrel{R}{\leftarrow} \mathbb{Z}_q\\ \quad X' \leftarrow (X/\mathtt{PW})^{s_b}, Y \leftarrow g^{s_0}\\ \text{output } b' \leftarrow \mathcal{A}(\mathtt{guess}, s, X', Y, \mathtt{PW})\end{array}$$

The PCDDH-assumption states that the advantage of $\mathcal{A}$ in deciding the PCDDH-problem, with respect to the dictionary $\mathcal{D}$, in reasonable time, is essentially $1/N$:

$$\mathsf{Adv}^{\mathsf{pcddh}}_{\mathcal{D}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{pcddh}}_1(\mathcal{A},\mathcal{D}) = 1] - \Pr[\mathbf{Exp}^{\mathsf{pcddh}}_0(\mathcal{A},\mathcal{D}) = 1] \leq \tfrac{1}{N} + \mathsf{negl}().$$

We insist on the fact that for the problem to be hard, the elements in the dictionary $\mathcal{D}$ must be independently drawn from $\mathbb{G}$. Namely, the relative discrete logarithms, of any quotient of any pair, in basis g must be hard to compute. Note that the computational variant, where the goal is to compute $X' = (X/\mathtt{PW})^{s_0}$, given Y and $\mathtt{PW}$, for a chosen X, with probability greater than $1/N$, holds under the CDH-assumption. Since the passwords are independent and random elements in $\mathcal{G}$, the adversary has to guess the password. Even without formal proof, such a decisional assumption seems reasonable under the DDH-one.

3.4 Security Result

Granted the PCDDH-assumption, one can prove the security of this protocol, in a strong sense, since the freshness notion is much more general than usual: it of course covers the forward-secrecy, by keeping a session as fresh even after a corruption of the parties. But even after corruptions, sessions where the adversary only forwards messages are also considered fresh. The security proof of our new GPAKE scheme can be found in the Appendix A.

Theorem 6 (Security). *Let us consider protocol from Figure 1 over a group of prime order q, where $\mathcal{D}$ is a dictionary of size N. Let us consider an adversary $\mathcal{A}$ that is able to initiate concurrent executions of the protocol, and to corrupt any party (in a non-adaptive way, i.e. one cannot corrupt a player in the middle of a session, but before the session starts only). If $\mathcal{A}$ makes less than q_{send} sessions with the parties, under the* CDH, DDH, *and* PCDDH *assumptions, where q_{send} is the number of* Send*-queries of $\mathcal{A}$, we have,*

$$\mathsf{Adv}^{\mathsf{ake}}(\mathcal{A}) \leq \frac{12 q_{\mathsf{send}}}{N} + \mathsf{negl}().$$

One can note that Execute-queries have no impact in the above result. Information leaked during such session is indeed negligible.

4 Adding Client Anonymity

In many applications, anonymity is a crucial property to be satisfied, otherwise all the connections of the clients can be logged and then analyzed to profile the users.

For privacy reasons, a client may want to make his connections anonymous and unlinkable. One way is first to use different gateways. But the authentication server is unique, and thus one cannot use a different one for each connection. Whereas the server is the only party able to authorize, or not, the connection, we will try to hide the client identity to it.

To this aim, we can see the server as a virtual dynamic database: for each authorization request, the server builds the answers for all the possible clients, but the gateway gets the one related to the actual client: the gateway and the server can run a PIR protocol [15, 16], so that the server does not learn anything about the client. Note that using an SPIR scheme [20] additionally provides privacy to the server, since the gateway will not learn more than the value it is interested in.

Recall that a PIR is a communication-efficient primitive which allows a user to retrieve a string in an n-string database without revealing anything about the index of the data the user is querying. An SPIR is a PIR, which satisfies the additional property of database privacy.

Several PIR have been proposed in the literature: with information-theoretic privacy for the user and/or the database, or with computational privacy only [14]. They may use one, or

several duplicated databases [19]. Recently, [8] proposed an efficient SPIR scheme based on a new homomorphic public-key cryptosystem which reduces considerably the complexity of existing schemes.

The nice feature of our new GPAKE is the efficient implementation with any SPIR. Namely, any good SPIR scheme can be applied with our construction, and thus we do not need to choose, nor to describe, a specific scheme, but just to use the query of a "black-box" system. We will show that we can reduce considerably the size of the "virtual" database, which would have a practical impact, whatever actual PIR scheme we use.

With our construction, each client owns a password indexed by i, which index is the secret of the gateway. The server manages a database of size n, the number of clients, which contains all the passwords for each client.

A trivial way to introduce anonymity to the protocols (ours, and [1]) using an SPIR is to dynamically generate a database, for each session, as follows: upon reception of a Send-query, with input $X^\star$, the server computes the answers for each message $(C_i, G, X^\star)$, and thus for all the possible clients C_i, since it does not know which one is interacting with the gateway G: the dynamic database consists of all the blocks $B_i = (g^{s_i}, (X^\star/\mathtt{PW}_i)^{s_i}, \Pi_i)$. Then, the gateway runs the SPIR protocol to get the correct B_i, while preserving the anonymity of the client. This transformation is quite generic, and one can note that the computational cost for the server for building the database is quite huge, essentially because of the multiple proofs Π_i.

We can easily improve efficiency in our case, by computing once (even pre-computing), and sending $h = g^s$ and Π_1, and then the dynamic database only consists of the n entries $B_i = (X^\star/\mathtt{PW}_i)^s$. This considerably improves the computational cost, and the storage space.

5 Conclusion

In this paper, we first strengthen the security model for password-based authenticated key exchange: we applied it to the gateway-based setting, but it is also relevant in the classical two-party scenario. We then design a new gateway-based key exchange protocol: a practical application is the establishment of a secure channel between a client and an application server with the help of an authentication server. The application server does not learn anything about the authentication material, and the authentication server does not learn anything about the session key. We focus on the password-based authentication approach, which is a quite practical one. Eventually, we address anonymity: the server does not need to be aware of the actual client. It should just check whether the identity and the password match (are in the database). Our new protocol can be efficiently interfaced with any PIR protocol to achieve client-anonymity.

Acknowledgment

We would like to thank the anonymous referees for their fruitful comments. This work has been partially supported by the French ANR-07-SESU-008-01 PAMPA Project.

References

1. Michel Abdalla, Olivier Chevassut, Pierre-Alain Fouque, and David Pointcheval. A simple threshold authenticated key exchange from short secrets. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 566–584. Springer, December 2005.
2. Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, January 2005.
3. Michel Abdalla, Malika Izabachène, and David Pointcheval. Anonymous and transparent gateway-based password-authenticated key exchange. In Matthew K. Franklin, Lucas Chi Kwong Hui, and Duncan S. Wong, editors, *Cryptology and Network Security, 7th International Conference, CANS 2008*, LNCS. Springer, 2008. Full version available on http://www.di.ens.fr/~pointche/pub.html.

4. Michel Abdalla and David Pointcheval. Interactive Diffie-Hellman assumptions with applications to password-based authentication. In Andrew Patrick and Moti Yung, editors, *FC 2005*, volume 3570 of *LNCS*, pages 341–356. Springer, February / March 2005.
5. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, May 2000.
6. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.
7. 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. IEEE Computer Society Press, May 1992.
8. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, February 2005.
9. Colin Boyd, Paul Montague, and Khanh Quoc Nguyen. Elliptic curve based password authenticated key exchange protocols. In *ACISP 01*, volume 2119 of *LNCS*, pages 487–501. Springer, 2001.
10. Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, May 2000.
11. Julien Bringer, Hervé Chabanne, David Pointcheval, and Qiang Tang. Extended private information retrieval and its application in biometrics authentications. In Feng Bao, San Ling, Tatsuaki Okamoto, Huaxiong Wang, and Chaoping Xing, editors, *CANS 07*, volume 4856 of *LNCS*, pages 175–193. Springer, December 2007.
12. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
13. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, May 2005.
14. Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *STOC*, pages 304–313, 1997.
15. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.
16. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
17. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, August 1987.
18. David P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997)*, pages 248–255, Cambridge, MA, USA, June 18–20, 1997. IEEE Computer Society.
19. Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997.
20. Laura Lincoln. *Symmetric Private Information Retrieval via Homomorphic Probabilistic Encryption*. PhD thesis, Rochester Institute of Technology, 2006. `http://www.cs.rit.edu/7Elbl6598/thesis/Lincoln_full_Dobument.pdf`.
21. Stefan Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Workshop on Security Protocols*, École Normale Supérieure, 1997.
22. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, May 1996.
23. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.
24. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, August 1990.
25. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.
26. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, May 1997.

A Security Proof

In this proof, we are interested in the event S_n which occurs when the adversary correctly guesses the bit b involved in the Test-query, that is, $b' = b$. To do so, we consider a sequence of games, which are slightly modified to reduce an attack into an algorithm against a computationally hard problem. We will denote by Δ_n the distance between two consecutive games $\mathbf{G}_n$ and $\mathbf{G}_{n+1}$.

We start with the real attack game. Note that even if the channel between the gateways and the server are private, we will simulate the communications. The adversary can record them and check later their validity, when it has corrupted a gateway.

Game $\mathbf{G}_0$: This is the real protocol, in the random-oracle model. By definition, we have:

$$\mathsf{Adv}^{\mathsf{ake}}_{\mathrm{GPAKE}}(\mathcal{A}) = 2\Pr[\mathsf{S}_0] - 1.$$

Game $\mathbf{G}_1$: In this game, we simulate the hash functions (random oracles) $\mathcal{G}$, $\mathcal{H}_1$ and $\mathcal{H}_2$, but also two additional hash functions $\mathcal{H}'_1, \mathcal{H}'_2 : \mathcal{U}^2 \times \mathbb{G}^2 \to \{0,1\}^\ell$, that will appear later in the security analysis (see Figure 2). We also simulate Send, and Execute-queries, as the players would proceed according to our new GPAKE protocol, and the Reveal and Test-queries as the challenger is supposed to do according to the security model (see Figure 3). From this simulation, we easily see that this game is perfectly indistinguishable from the real protocol.

Game $\mathbf{G}_2$: To proceed to an easier analysis, we remove games in which some collisions appear:

- collisions on the output of $\mathcal{G}$
- collisions on the output of $\mathcal{H}_1$ and $\mathcal{H}_2$
- collisions on the partial transcripts $(C, G, X^\star, \overline{Y})$

$$\Delta_2 \leq \frac{q^2_{\mathsf{session}} + q^2_{\mathcal{G}}}{2q} + \frac{q^2_{\mathcal{H}}}{2^\ell} = \mathsf{negl}(),$$

where q_{session} is the number of sessions, and $q_{\mathcal{G}}$, $q_{\mathcal{H}}$ are the number of queries to the oracles $\mathcal{G}$ and $\mathcal{H}_1, \mathcal{H}_2$ respectively.

Game $\mathbf{G}_3$: In this game, we consider trivial attacks in which the adversary tries to manufacture an authenticator AuthG. In a first step, we show that if there is no $\mathcal{H}_2$-query associated to AuthG, then the attack fails with overwhelming probability. Note however that if such a query exists, it is unique, since we have removed collisions from $\mathcal{H}_2$ (and $\mathcal{H}_1$) in $\mathbf{G}_2$. Furthermore, we modify the simulation of the client, when nobody is corrupted but the client receives a non-oracle-generated message: it rejects the authenticator.

▶**Rule C2**$^{(3)}$
- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \neg\mathsf{OG}(C, 4)$, reject.
- Lookup in $\Lambda_{\mathcal{H}}$ for K such that $\mathsf{AuthG}' = \mathcal{H}_2(C, G, X^\star, \overline{Y}, K)$.
  - if there is not, then reject.
  - If Π_1 and Π_2 are valid and $K = \overline{Y}^x$, then compute $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$, else reject.

$\mathbf{G}_3$ is identical to $\mathbf{G}_2$ except if a valid authenticator has been rejected:

- BadReject-NoAskH: the hash query has not been asked, but the authenticator is valid;
- BadReject-Uncorrupted: nobody is corrupted, but the adversary manages to build a valid authenticator (without the help of the authentication server, since it cannot ask any query not intercept any communication from the server, because of the private channel).

Hash oracles	
	For a hash-query $\mathcal{H}_n(q)$ (resp. $\mathcal{H}'_n(q)$), such that a record (n, q, r) appears in $\Lambda_{\mathcal{H}}$ (resp. $\Lambda_{\mathcal{H}'}$), the answer is r. Otherwise one chooses a random element $r \in \{0,1\}^\ell$, answers with it, and adds the record (n, q, r) to $\Lambda_{\mathcal{H}}$ (resp. $\Lambda_{\mathcal{H}'}$).
	For a hash-query $\mathcal{G}(q)$ such that a record (q, r) appears in $\Lambda_{\mathcal{G}}$, the answer is r. Otherwise the answer r is chosen at random in $\mathbb{G}$. The record (q, r) is added to $\Lambda_{\mathcal{G}}$.

Fig. 2. Simulation of the hash function in our new GPAKE protocol

Send-queries to C

We answer to the Send-queries to a C-instance as follows:

- A $\mathsf{Send}(C;\mathsf{INIT})$-query is processed according to the following rule:
  - ▶**Rule C1**$^{(1)}$
    - Choose a random exponent $x \in \mathbb{Z}_q$, compute $X = g^x$ and $X^\star = X \times \mathtt{PW}$.

  Then the query is answered with $(C, X^\star)$ and the instance goes to an expecting state.
- A $\mathsf{Send}(C, G; h, \overline{Y}, \mathsf{AuthG}, \Pi_1, \Pi_2)$-query is processed according to the following rule:
  - ▶**Rule C2**$^{(1)}$
    - Compute $K = \overline{Y}^x$ and $\mathsf{AuthG}' = \mathcal{H}_2(C, G, X^\star, \overline{Y}, K)$.
    - If Π_1 and Π_2 are valid and $\mathsf{AuthG} = \mathsf{AuthG}'$,
      then compute $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$, else reject.

  Then fresh_C is defined considering the previous definition 3. Finally, C accepts if everything is correct, and terminates in any case.
- All the other cases are ignored.

Send-queries to G

We answer to the Send-queries to a G-instance as follows:

- A $\mathsf{Send}(G, C; C, X^\star)$-query is simply processed by forwarding back $(C, G, X^\star)$.
- A $\mathsf{Send}(G, S; \overline{X}, h, \Pi_1)$-query is processed according to the following rules:
  - ▶**Rule G1**$^{(1)}$
    - Check whether Π_1 is valid or not.
    - Choose random $y \in \mathbb{Z}_q$. Compute $\overline{Y} = h^y$ and $K = \overline{X}^y$.
    - Construct $\Pi_2 = \mathtt{NIZKPDL}(X^\star; h, \overline{Y})$, granted y.
  - ▶**Rule G2**$^{(1)}$
    - Compute $\mathsf{AuthG} = \mathcal{H}_2(C, G, X^\star, \overline{Y}, K)$ and $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$

  Then fresh_G is defined considering the definition 3. Then, G accepts and terminates.
  The query is answered with $(G, h, \overline{Y}, \mathsf{AuthG}, \Pi_1, \Pi_2)$.
- All the other cases are ignored.

Send-queries to S

We answer to the Send-queries to an S-instance as follows:

- A $\mathsf{Send}(S, G; C, G, X^\star)$-query is processed according to the following rules:
  - ▶**Rule S**$^{(1)}$
    - Choose a random $s \in \mathbb{Z}_q$ and compute $h = g^s$.
    - Compute $\overline{X} = (X^\star/\mathtt{PW})^s$, where $\mathtt{PW}$ is C's password.
    - Construct $\Pi_1 = \mathtt{NIZKPDL}(X^\star; g, h)$, granted s.

  The query is answered with $(\overline{X}, h, \Pi_1)$
- All the other cases are ignored.

Other queries

An $\mathsf{Execute}(C_i^r, G_j^s, S^t)$ is processed using successively the simulation of Send-queries as follows:

$(C, X^\star) \leftarrow \mathsf{Send}(C; \mathsf{INIT})$, using $\mathbf{EC1}^{(1)}$ instead of $\mathbf{C1}^{(1)}$;
$(C, G, X^\star) \leftarrow \mathsf{Send}(G, C; C, X^\star)$;
$(\overline{X}, h, \Pi_1) \leftarrow \mathsf{Send}(S, G; C, G, X^\star)$, using $\mathbf{ES}^{(1)}$ instead of $\mathbf{S}^{(1)}$;
$(G, h, \overline{Y}, \mathsf{AuthG}, \Pi_1, \Pi_2) \leftarrow \mathsf{Send}(G, S; \overline{X}, h, \Pi_1)$,
using $\mathbf{EG1}^{(1)}$ and $\mathbf{EG2}^{(1)}$ instead of $\mathbf{G1}^{(1)}$ and $\mathbf{G2}^{(1)}$ respectively;
and $\mathsf{Send}(C, G; h, \overline{Y}, \mathsf{AuthG}, \Pi_1, \Pi_2)$, using $\mathbf{EC2}^{(1)}$ instead of $\mathbf{C2}^{(1)}$.

Then, return the transcript $(C, G, X^\star), (\overline{X}, h, \Pi_1), (G, h, \mathsf{AuthG}, \Pi_1, \Pi_2)$.

A $\mathsf{Reveal}(U)$-query returns the session key sk_U, if the instance has already accepted and if the instance has not been tested yet. And it returns $\perp$ otherwise.

A $\mathsf{Test}(U)$-query returns always the same random $sk \in SK$, if $b = 0$. Otherwise it returns $\mathsf{Reveal}(U)$.

Fig. 3. Simulation of our new GPAKE protocol

Then, $\Delta_3 \leq \Pr[\mathsf{BadReject\text{-}Uncorrupted}_3] + \Pr[\mathsf{BadReject\text{-}NoAskH}_3]$. We deal with the latter event BadReject-Uncorrupted later. For the former case, it is easy to see that:

$$\Pr[\mathsf{BadReject\text{-}NoAskH}_3] \leq \frac{q_{\mathsf{send}}}{2^l} = \mathsf{negl}(),$$

where q_{send} is the number of Send-queries.

Note that such a BadReject-Uncorrupted-event may happen for one password for each session, since a tuple $(g, X^\star/\mathtt{PW}, \overline{Y}, K)$ for the K involved in the $\mathcal{H}_2$-query may be a CDH-tuple for at most one value of PW. However, since our simulation still uses the valid password, we cannot claim that the probability is bounded by q_{send}/N, yet. The view of the adversary may leak some information about the password, so that it guesses it more easily. The next games aim at achieving this goal: simulate everything without using the password, and thus providing no bias to the adversary.

Game $\mathbf{G}_4$: In this game, we consider only attacks generated via Execute-queries. In such a case, we modify the simulation of Execute-queries by replacing oracles $\mathcal{H}_1$ and $\mathcal{H}_2$ by the private ones $\mathcal{H}_1'$ and $\mathcal{H}_2'$. Then, in such cases, we do not need to compute the Diffie-Hellman key K, since both the authenticator and the session key are completely independent from it. Let see how we simulate Execute-queries in this game:

▶**Rule EG1**$^{(4)}$
> Choose random $y \in \mathbb{Z}_q$, compute $\overline{Y} = h^y$.
> Simulate $\Pi_2 = \mathtt{NIZKPDL}(X^\star; h, \overline{Y})$, without using y.

▶**Rule EG2**$^{(4)}$
> Compute $\mathsf{AuthG} = \mathcal{H}_2{}'(C, G, X^\star, \overline{Y})$ and $sk = \mathcal{H}_1{}'(C, G, X^\star, \overline{Y})$.

▶**Rule EC2**$^{(4)}$
> Compute $sk = \mathcal{H}'_1(C, G, X^\star, \overline{Y})$.

Games $\mathbf{G}_4$ and $\mathbf{G}_3$ are indistinguishable unless $\mathcal{A}$ queries hash function $\mathcal{H}_1$ or $\mathcal{H}_2$ on some execution transcript $C\|G\|X^\star\|\overline{Y}^\star\|K$. As we restrict the analysis to attacks via Execute-queries, we call this event AskHE. For any event Ev, we have

$$|\Pr[\mathsf{Ev}_4] - \Pr[\mathsf{Ev}_3]| \leq \Pr[\mathsf{AskHE}_4].$$

If event AskHE_4 occurs, this means that, for some Execute-transcripts, $\mathcal{A}$ can distinguish whether the simulator has used the private oracle or the public one. In other words, the tuple $(X^\star, \overline{Y}, K)$, with $K = \mathsf{CDH}(X^\star/\mathtt{PW}, \overline{Y}))$ lies in the list $\Lambda_{\mathcal{H}}$, which means that $\mathcal{A}$ can break the CDH-problem.

To show that, we consider an auxiliary game, in which we do not change anything in the view of the adversary, but we modify the simulation when $\mathcal{A}$ interacts with the protocol via Execute-queries. It is worth noting that the simulator knows the password, which enables us to simulate an INIT-query with $X = Ag^\alpha$ and $\overline{Y} = Bg^\beta$, where $A = g^a$ and $B = g^b$ are two random inputs for the CDH-problem.

Game $\mathbf{G}_{4.1}$:

▶**Rule EC1**$^{(4.1)}$
> Choose a random exponent $\alpha \in \mathbb{Z}_q$,
> compute $X = Ag^\alpha$ and $X^\star = X \times \mathtt{PW}$.

▶**Rule EG1**$^{(4.1)}$
> Choose a random $\beta \in \mathbb{Z}_q$
> and compute $\overline{Y} = Bg^\beta$.
> Simulate $\Pi_2 = \mathtt{NIZKPDL}(X^\star; h, \overline{Y})$, without using y.

Thus, if event AskHE occurs, it means that $\mathcal{A}$ can extract the CDH-value of A and B in $\Lambda_{\mathcal{H}}$:

$$K = \mathsf{CDH}(Ag^{\alpha}, Bg^{\beta}) = \mathsf{CDH}(A, B) \times B^{\alpha} A^{\beta} g^{\alpha\beta}.$$

Then, we obtain $\mathsf{CDH}(A, B) = K/B^{\alpha} A^{\beta} g^{\alpha\beta}$:

$$\Delta_4 \leq \Pr[\mathsf{AskHE}_4] \leq q_{\mathcal{H}} \times \mathsf{Succ}^{\mathsf{cdh}}(t') = \mathsf{negl}(),$$

where $t' = t + (2q_{\mathsf{execute}} + 3)\tau$, and τ the time for one exponentiation in the group and where q_{execute} is the number of $\mathsf{Execute}$-queries.

Note that in this game, $\mathsf{Execute}$-queries do not involve the password, for the client and the gateway, since they do not need to compute the authenticator nor the session key.

Game $\mathbf{G}_5$: However, the server still needs the password to compute correctly h and $\overline{X}$. We now simulate it without using the password:

▶**Rule ES**$^{(5)}$

Choose a random $s \in \mathbb{Z}_q$ and compute $h = g^s$.
Choose a random $t \in \mathbb{Z}_q$ and compute $\overline{X} = g^t$.
Simulate $\Pi_1 = \mathtt{NIZKPDL}(X^{\star}; g, h)$, without using s.

Games $\mathbf{G}_5$ and $\mathbf{G}_4$ are identical except in the simulation of the server. Let us prove that the difference cannot be detected unless one can break the DDH-assumption.

Game $\mathbf{G}_{5.1}$: Let us be given a CDH-triple (A, B, C):

▶**Rule EC1**$^{(5.1)}$

Choose random $x_1, x_2 \in \mathbb{Z}_q$, and compute $X = g^{x_1} A^{x_2}$ and $X^{\star} = X \times \mathtt{PW}$.

▶**Rule ES**$^{(5.1)}$

Choose random $y_1, y_2 \in \mathbb{Z}_q$, and compute $h = g^{y_1} B^{y_2}$,
and $\overline{X} = C^{x_2 y_2} A^{x_2 y_1} B^{x_1 y_2} g^{x_1 x_2}$.
Simulate $\Pi_1 = \mathtt{NIZKPDL}(X^{\star}; g, h)$.

It is easy to see that this simulation is perfectly identical to the one in game $\mathbf{G}_4$.

Game $\mathbf{G}_{5.2}$: We now just modify the input (A, B, C), with a random triple: then the simulation is perfectly identical to the one in game $\mathbf{G}_4$, then $\Delta_5 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t') = \mathsf{negl}()$, where $t' = t + 8q_{\mathsf{execute}}\tau$, and τ the time for one exponentiation in the group and where q_{execute} is the number of $\mathsf{Execute}$-queries.

Game $\mathbf{G}_6$: In this game, we restrict our analysis to games in which the adversary simply forwards all the flows via Send-queries . They are thus actually passive executions, similar to $\mathsf{Execute}$-queries, but we do not know, from the beginning if the whole session will consist of forwarded messages or not. We want to show that the adversary cannot have any idea about the session key, unless it can break the CDH-problem. To show that, we make the simulator to use private functions $\mathcal{H}'_1$ and $\mathcal{H}'_2$ instead of $\mathcal{H}_1$ and $\mathcal{H}_2$, without the value K, as above.

▶**Rule G1**$^{(6)}$

Check whether Π_1 is valid or not.
Then,

- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \mathsf{OG}(G, 1)$,
  choose random $y \in \mathbb{Z}_q$ and compute $\overline{Y} = h^y$.
- else, choose random $y \in \mathbb{Z}_q$. Compute $\overline{Y} = h^y$ and $K = \overline{X}^y$.

Simulate $\Pi_2 = \mathtt{NIZKPDL}(X^{\star}; h, \overline{Y})$, without using y.

►**Rule G2**$^{(6)}$
- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \mathsf{OG}(G,1)$, compute
  $\mathsf{AuthG} = \mathcal{H}_2{}'(C,G,X^\star,\overline{Y})$ and $sk = \mathcal{H}_1{}'(C,G,X^\star,\overline{Y})$.
- else, compute
  $\mathsf{AuthG} = \mathcal{H}_2(C,G,X^\star,\overline{Y},K)$ and $sk = \mathcal{H}_1(C,G,X^\star,\overline{Y},K)$

►**Rule C2**$^{(6)}$
- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \neg\mathsf{OG}(C,4)$, reject.
- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \mathsf{OG}(G,1)$,
  compute $\mathsf{AuthG}' = \mathcal{H}'_2(C,G,X^\star,\overline{Y})$.
  If Π_1 and Π_2 are valid, and $\mathsf{AuthG}' = \mathsf{AuthG}$,
  then compute $sk = \mathcal{H}'_1(C,G,X^\star,\overline{Y})$, else reject.
- Lookup in $\Lambda_{\mathcal{H}}$ for K such that $\mathsf{AuthG}' = \mathcal{H}_2(C,G,X^\star,\overline{Y},K)$.
  - if there is not, then reject.
  - If Π_1 and Π_2 are valid and $K = \overline{Y}^x$,
    then compute $sk = \mathcal{H}_1(C,G,X^\star,\overline{Y},K)$, else reject.

The probability of any event in game $\mathbf{G}_6$ and $\mathbf{G}_5$ is the same unless the adversary asks $\mathcal{H}_1$ or $\mathcal{H}_2$ on some particular inputs $C\|G\|X^\star\|\overline{Y}\|K$, where K is the Diffie-Hellman value of $X^\star/\mathtt{PW}$ and $\overline{Y}$. We call this event AskHF_6, since we restrict our analysis to the passive case: simple forwarding of messages. For any event Ev, we have

$$|\Pr[\mathsf{Ev}_6] - \Pr[\mathsf{Ev}_5]| \leq \Pr[\mathsf{AskHF}_6].$$

Exactly as above in game $\mathbf{G}_4$, we can show that

$$\Delta_6 \leq \Pr[\mathsf{AskHF}_6] \leq q_{\mathcal{H}} \times \mathsf{Succ}^{\mathsf{cdh}}(t') = \mathsf{negl}(),$$

where $t' = t + (2q_{\mathsf{send}} + 3)\tau$, and τ the time for one exponentiation in the group and where q_{send} is the number of Send-queries.

Game $\mathbf{G}_7$: We now consider the case where the gateway is corrupted, and then the adversary can impersonate the server with respect to the gateway (we have assumed a symmetric authentication). But since it does not know the password, we will make the client to reject. It is actually the same situation as if the adversary directly impersonates the gateway, without asking any help to the server ($\neg\mathsf{OGA}(S)$, where OGA means that a query has been asked to the oracle, and the input was oracle generated):

►**Rule C2**$^{(7)}$
- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \neg\mathsf{OG}(C,4)$, reject.
- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \mathsf{OG}(G,1)$,
  compute $\mathsf{AuthG}' = \mathcal{H}'_2(C,G,X^\star,\overline{Y})$.
  If Π_1 and Π_2 are valid, and $\mathsf{AuthG}' = \mathsf{AuthG}$,
  then compute $sk = \mathcal{H}'_1(C,G,X^\star,\overline{Y})$, else reject.
- if $\neg\mathsf{corrupt}_C \wedge (\neg\mathsf{OGA}(S) \vee \neg\mathsf{OG}(G,3))$, reject.
- Lookup in $\Lambda_{\mathcal{H}}$ for K such that $\mathsf{AuthG}' = \mathcal{H}_2(C,G,X^\star,\overline{Y},K)$.
  - if there is not, then reject.
  - If Π_1 and Π_2 are valid and $K = \overline{Y}^x$,
    then compute $sk = \mathcal{H}_1(C,G,X^\star,\overline{Y},K)$, else reject.

As for the analysis of the game $\mathbf{G}_3$, $\mathbf{G}_7$ is identical to $\mathbf{G}_6$ unless the adversary tries to play instead of the server, and correctly cancel the password. Then, we will reject a valid authenticator. But this is very unlikely since the adversary does not know the correct password (the client is not corrupted): For any event Ev, we have

$$|\Pr[\mathsf{Ev}_7] - \Pr[\mathsf{Ev}_6]| \leq \Delta_7 \leq \Pr[\mathsf{BadReject\text{-}C\text{-}Uncorrupted}_7].$$

Note that such a BadReject-C-Uncorrupted-event may happen for one password for each session, since a tuple $(g, X^\star/\mathtt{PW}, h, \overline{X})$ output by the adversary may be a CDH-tuple for at most one value of PW. However, since our simulation still uses the valid password, we cannot claim that the probability is bounded by q_{send}/N, where q_{send} is the number of Send-queries. The next games will lead to the conclusion.

Game $\mathbf{G}_8$: We now consider attacks in which $\mathcal{A}$ directly interacts with the server (the gateway may be corrupted). Note that when simulating an INIT-query to the client, we do not know yet whether the generated $X^\star$ will be forwarded to S or not. The adversary can send a different value. The client would likely reject, but some information about the actual password may be leaked by the server, since the latter uses the password. We will however show that no valuable information about the password actually leaks, since our simulation of the server will not use the password anymore.

▶**Rule C1**$^{(8)}$
- if $\neg\mathsf{corrupt}_C$, choose a random exponent $x^\star \in \mathbb{Z}_q$, compute $X^\star = g^{x^\star}$.
- else, choose a random exponent $x \in \mathbb{Z}_q$, compute $X = g^x$ and $X^\star = X \times \mathtt{PW}$.

▶**Rule S**$^{(8)}$

Choose a random $s \in \mathbb{Z}_q$ and compute $h = g^s$.
Then,
- if $\neg\mathsf{corrupt}_C$, choose a random $t \in \mathbb{Z}_q$, compute $\overline{X} = g^t$.
- else compute $\overline{X} = (X^\star/\mathtt{PW}_i)^s$, where $\mathtt{PW}_i$ is C_i's password.

Simulate $\Pi_1 = \mathtt{NIZKPDL}(X^\star; g, h)$.

▶**Rule C2**$^{(8)}$
- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \neg\mathsf{OG}(C, 4)$, reject.
- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \mathsf{OG}(G, 1)$,
  compute $\mathsf{AuthG}' = \mathcal{H}'_2(C, G, X^\star, \overline{Y})$.
  If Π_1 and Π_2 are valid, and $\mathsf{AuthG}' = \mathsf{AuthG}$,
  then compute $sk = \mathcal{H}'_1(C, G, X^\star, \overline{Y})$, else reject.
- if $\neg\mathsf{corrupt}_C \wedge (\neg\mathsf{OGA}(S) \vee \neg\mathsf{OG}(G, 3))$, reject.
- Lookup in $\Lambda_{\mathcal{H}}$ for K such that $\mathsf{AuthG} = \mathcal{H}_2(C, G, X^\star, \overline{Y}, K)$.
  - if there is not, then reject.
  - if $\neg\mathsf{corrupt}_C$, if Π_1 and Π_2 are valid and $\overline{Y}^t = K^s$,
    then compute $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$, else reject.
  - else, if Π_1 and Π_2 are valid and $K = \overline{Y}^x$,
    then compute $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$, else reject.

Games $\mathbf{G}_8$ and $\mathbf{G}_7$ are identical except when the client is not corrupted:

- $X^\star$ is simulated without using the password (but it makes no difference),
- then the server simulates h and $\overline{X}$ independently, without using the password either.

Thereafter, the behavior of the client is modified properly. We want to prove that no matter who provides the value $X^\star$ involved in Send-query comes, the adversary cannot distinguish the simulation of the server in the two games, except with a negligible advantage.

To prove this result, we use a hybrid argument to show that if the above advantage is not negligible, then we can break the PCDDH-problem. We construct a sequence of $q_{\mathsf{send}} + 1$ games denoted by Hyb_k, for $0 \leq k \leq q_{\mathsf{send}}$, where q_{send} is the number of Send-queries. In each game Hyb_k, the i-th session (ordered according to the Send-query) is processed as follows:

- If $(i \leq k)$, then process Send-queries as in $\mathbf{G}_8$.

– else, process Send-queries as in $\mathbf{G}_7$.

Let us now define an intermediate game G_k, which is similar to Hyb_k, excepted:

►**Rule S**$^{(8,k)}$
- if $i \leq k$, proceed as in $\mathbf{G}_8$.
- if $i = k+1$, send $(\mathcal{D}, X^\star)$ as input to the PCDDH-challenger, that sends back $(\overline{X}, h, \mathtt{PW})$. Return $\overline{X}$ and h defined above, by the PCDDH-challenger output.
  Simulate $\Pi_1 = \mathtt{NIZKPDL}(X^\star; g, h)$.
- else, proceed as in $\mathbf{G}_7$, with $\mathtt{PW}$.

►**Rule C2**$^{(8,k)}$
- if $i \leq k$, proceed as in $\mathbf{G}_8$.
- if $i = k+1$,
  - if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \neg\mathsf{OG}(C, 4)$, reject.
  - if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \mathsf{OG}(G, 1)$,
    compute $\mathsf{AuthG}' = \mathcal{H}'_2(C, G, X^\star, \overline{Y})$.
    If Π_1 and Π_2 are valid, and $\mathsf{AuthG}' = \mathsf{AuthG}$,
    then compute $sk = \mathcal{H}'_1(C, G, X^\star, \overline{Y})$, else reject.
  - if $\neg\mathsf{corrupt}_C \wedge (\neg\mathsf{OGA}(S) \vee \neg\mathsf{OG}(G, 3))$, reject.
  - Lookup in $\Lambda_{\mathcal{H}}$ for K such that $\mathsf{AuthG} = \mathcal{H}_2(C, G, X^\star, \overline{Y}, K)$.
    - if there is not, then reject.
    - if $\neg\mathsf{corrupt}_C$,
      - if Π_1 or Π_2 is not valid, reject;
      - extract y from Π_2 (with one rewind, or fail);
      - if $K = \overline{X}^y$,
        then compute $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$, else reject.
    - else, if Π_1 and Π_2 are valid and $K = \overline{Y}^x$, then compute $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$, else reject.
- else proceed as in $\mathbf{G}_7$.

$\mathcal{A}'$ returns its guess $d' = (b = b')$ for d, where b' is the guess of $\mathcal{A}$ for the bit b, since we consider S, but it could be $d' = 1$ if and only if any observable event Ev happens.

Actually, this game is either identical to experiment Hyb_k if $d = 0$, or to experiment Hyb_{k+1} otherwise, granted the extraction of y:

– If $d = 0$, $(g, X^\star/\mathtt{PW}, h, \overline{X})$ is a CDH-tuple, thus the server is simulated as in $\mathbf{G}_7$. Then, for the client, testing $K = \overline{X}^y$ is equivalent to test $K = \overline{Y}^x$;
– If $d = 1$, it is also easy to see that the server is simulated as in $\mathbf{G}_8$, since $(g, X^\star/\mathtt{PW}, h, \overline{X})$ is a random tuple. For the client, we just verify that the adversary honestly computed $\overline{Y}$ and K, with a common exponent y.

Thus, games Hyb_k and Hyb_{k+1} are indistinguishable unless $\mathcal{A}$ has a non-negligible advantage in guessing b correctly (or in distinguishing any observable event Ev). But in such a case, $\mathcal{A}'$ guesses d with a non-negligible advantage:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{pcddh}}_{\mathbb{G}}(t) \geq \mathsf{Adv}^{\mathsf{pcddh}}_{\mathbb{G}}(\mathcal{A}') &= \Pr[d' = 1 | d = 1] - \Pr[d' = 1 | d = 0] \\ &= \Pr_{k+1}[\mathsf{Ev}] - \Pr_k[\mathsf{Ev}].\end{aligned}$$

As a consequence, under the PCDDH-assumption,

$$\Delta_8 \leq \frac{q_{\mathsf{send}}}{N} + \mathsf{negl}().$$

One should note that the ordering is done according to the Send-queries. Then, one may wonder if the proof applies to non-concurrent executions only, or also to concurrent executions.

Actually, the simulation of the flows before this query are the same in all the hybrid games, since corruptions are non-adaptive: the adversary cannot corrupt a party in the middle of an execution of protocol. Then, the analysis allows concurrent executions.

Game $\mathbf{G}_9$: In this game, we now deal with the sessions in which the client is corrupted, which means that $\mathcal{A}$ (but also the simulator) knows the password.

In this game, we replace the public oracles $\mathcal{H}_1$ and $\mathcal{H}_2$ by the private ones $\mathcal{H}'_1$ and $\mathcal{H}'_2$ in sessions which may be fresh:

- even if the client is corrupted, the adversary may let it play by itself. Then, if $\mathsf{OG}(G, 1)$, then the session may be fresh.
- however, if $\neg\mathsf{OG}(G, 1)$, we know that the session will not be fresh. But the protocol must guarantee that the client is able to verify the validity of the authenticator, whoever sends the authenticator.

Since we removed trivial cases in $\mathbf{G}_3$ (authenticator not asked to the hash function), then upon reception of $(\overline{Y}, \mathsf{AuthG})$, the client will extract the input $(C, G, X^\star, \overline{Y}, K)$ such that $\mathsf{AuthG} = \mathcal{H}_2(C, G, X^\star, \overline{Y}, K)$. Thus, if $\neg\mathsf{OG}(C, 4)$, we must verify that the client is able to reject transcripts for which $(g, X^\star/\mathtt{PW}, \overline{Y}, K)$ is not a CDH-tuple, when using the public oracles.

►**Rule G1**$^{(9)}$

Check whether Π_1 is valid or not.

- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G$ or $\mathsf{OG}(S, 2) \wedge \mathsf{OG}(G, 3)$,
  choose a random exponent $y \in \mathbb{Z}_q$ and compute $\overline{Y} = h^y$;
- else, choose a random exponent $y \in \mathbb{Z}_q$, compute $\overline{Y} = h^y$ and $K = \overline{X}^y$.

Simulate $\Pi_2 = \mathtt{NIZKPDL}(X^\star; h, \overline{Y})$.

►**Rule G2**$^{(9)}$

- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G$ or $\mathsf{OG}(S, 2) \wedge \mathsf{OG}(G, 3)$,
  compute $\mathsf{AuthG} = \mathcal{H}'_2(C, G, X^\star, \overline{Y})$ and $sk = \mathcal{H}'_1(C, G, X^\star, \overline{Y})$;
- else, compute $\mathsf{AuthG} = \mathcal{H}_2(C, G, X^\star, \overline{Y}, K)$
  and $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$.

►**Rule C2**$^{(9)}$

- if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G \wedge \neg\mathsf{OG}(C, 4)$, reject.
- if $\neg\mathsf{corrupt}_C \wedge (\neg\mathsf{OGA}(S) \vee \neg\mathsf{OG}(G, 3))$, reject.
- if $\mathsf{OG}(G, 3) \wedge \mathsf{OG}(C, 4)$, compute $\mathsf{AuthG}' = \mathcal{H}'_2(C, G, X^\star, \overline{Y})$.
  If Π_1 and Π_2 are valid, and $\mathsf{AuthG}' = \mathsf{AuthG}$,
  then compute $sk = \mathcal{H}'_1(C, G, X^\star, \overline{Y})$, else reject.
- Lookup in $\Lambda_{\mathcal{H}}$ for K such that $\mathsf{AuthG} = \mathcal{H}_2(C, G, X^\star, \overline{Y}, K)$.
  - if there is not, then reject.
  - if $\neg\mathsf{corrupt}_C$,
    if Π_1 and Π_2 are valid and $\overline{Y}^t \neq K^s$,
    then compute $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$, else reject.
  - else, if Π_1 and Π_2 are valid and $K = \overline{Y}^x$,
    then compute $sk = \mathcal{H}_1(C, G, X^\star, \overline{Y}, K)$, else reject.

The difference between $\mathbf{G}_9$ and $\mathbf{G}_8$ is that we replace $\mathcal{H}_1$ and $\mathcal{H}_2$ by $\mathcal{H}'_1$ and $\mathcal{H}'_2$ in many more sessions: in all the sessions in which the session key is not compromised (chosen by oracles). To show that this difference is negligible, we define below a sequence of hybrid experiments Hyb_k for $0 \leq k \leq q_{\mathsf{send}}$. For the i-th session, ordered according the INIT-query:

- if $i \leq k$, process Send-queries as in $\mathbf{G}_9$.
- else process Send-queries as in $\mathbf{G}_8$.

We want to show that the difference of the success of $\mathcal{A}$ in breaking the semantic security of the session key in Hyb_k and Hyb_{k+1} is negligible: the adversary asks $\mathcal{H}_1$ or $\mathcal{H}_2$ on some transcript $C\|G\|X^\star\|\overline{Y}\|K$ simulated using $\mathcal{H}'_1$ or $\mathcal{H}'_2$. The adversary has found the Diffie-Hellman value $K = \mathsf{CDH}(X, \overline{Y}) = \overline{Y}^x$. We call this event AskHA_9, since now some of the participants may be corrupted. For any event Ev, we have:

$$|\Pr_{k+1}[\mathsf{Ev}] - \Pr_k[\mathsf{Ev}]| \leq \Pr[\mathsf{AskHA}_9].$$

To upper-bound the probability of event AskHA_k, we reduce the adversary behavior in Hyb_k and Hyb_{k+1} to the behavior of an adversary, $\mathcal{A}'$ in the CDH-experiment. Let (A, B) be a CDH-input.

►**Rule C1**$^{(9,k)}$
- if $(i \neq k)$, process $\mathsf{Send}(C, \mathsf{INIT})$-query as in $\mathbf{G}_8$=$\mathbf{G}_9$.
- else, if $i = k$,
  - if $\neg\mathsf{corrupt}_C$, choose a random exponent $x^\star \in \mathbb{Z}_q$, compute $X^\star = g^{x^\star}$.
  - else if $\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G$, set $X = A$, and compute $X^\star = X \times \mathtt{PW}$.
  - else choose a random $x \in \mathbb{Z}_q$, compute $X = g^x$ and $X^\star = X \times \mathtt{PW}$.

►**Rule G1**$^{(9,k)}$
- if $i < k$, proceed as in $\mathbf{G}_9$.
- else, if $i = k$,
  - Check whether Π_1 is valid or not.
  - Then,
    - if $\neg\mathsf{corrupt}_C \wedge \neg\mathsf{corrupt}_G$ or $\mathsf{OG}(S, 2) \wedge \mathsf{OG}(G, 3)$, set $\overline{Y} = B$;
    - else, choose a random $y \in \mathbb{Z}_q$ and compute $\overline{Y} = h^y$ and $K = \overline{X}^y$.
  - Construct $\Pi_2 = \mathtt{NIZKPDL}(X^\star; h, \overline{Y})$.
- else proceed as in $\mathbf{G}_8$

If the event AskHA_k occurs, then we can extract the CDH-value of A and B in $\Lambda_{\mathcal{H}}$: $\Pr[\mathsf{AskHA}_k] \leq \mathsf{Succ}^{\mathsf{cdh}}(t)$. As a consequence,

$$\Delta_9 \leq q_{\mathsf{send}} \times \mathsf{Succ}^{\mathsf{cdh}}(t) = \mathsf{negl}().$$

In this final game, we can remark two things: first, the keys are computed using a private function when they are fresh. Therefore, the adversary cannot distinguish them from truly random keys.

$$\Pr[\mathsf{S}_8] = \frac{1}{2}.$$

Furthermore, the password is not used, except when the client is corrupted, we can thus choose it when needed only:

- when the client is corrupted;
- when the adversary concludes and outputs its guess b' for the bit b;

and then check whether events $\mathsf{BadReject\text{-}Uncorrupted}_8$ and $\mathsf{BadReject\text{-}C\text{-}Uncorrupted}_8$ happened or not. But as noted before, only one password per session can raise these events, then

$$\Pr[\mathsf{BadReject\text{-}Uncorrupted}_8] \leq \frac{q_{\mathsf{send}}}{N} \qquad \Pr[\mathsf{BadReject\text{-}C\text{-}Uncorrupted}_8] \leq \frac{q_{\mathsf{send}}}{N}.$$

$$\Delta_2 = \mathsf{negl}() \quad \Delta_3 = \Pr[\mathsf{BadReject\text{-}Uncorrupted}_3] + \mathsf{negl}() \quad \Delta_4 \leq \Delta_4 = \mathsf{negl}()$$

$$\Delta_5 = \mathsf{negl}() \quad \Delta_6 = \Pr[\mathsf{BadReject\text{-}C\text{-}Uncorrupted}_6] \quad \Delta_7 \leq \frac{q_{\mathsf{send}}}{N} + \mathsf{negl}() \quad \Delta_8 = \mathsf{negl}()$$

Therefore,

$$\Delta_6 = \Pr[\mathsf{BadReject\text{-}C\text{-}Uncorrupted}_6] \leq \frac{2q_{\mathsf{send}}}{N} + \mathsf{negl}()$$

$$\Delta_3 = \Pr[\mathsf{BadReject\text{-}Uncorrupted}_3] + \mathsf{negl}() \leq \frac{q_{\mathsf{send}}}{N} + \frac{3q_{\mathsf{send}}}{N} + \mathsf{negl}()$$

$$\mathsf{S}_0 \leq \frac{1}{2} + \frac{q_{\mathsf{send}}}{N} + \frac{5q_{\mathsf{send}}}{N} + \mathsf{negl}() \leq \frac{1}{2} + \frac{6q_{\mathsf{send}}}{N} + \mathsf{negl}(),$$

where q_{send} is the number of sessions where the adversary is active (*i.e.*plays via using Send-queries).